Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Sandro Ribolla

Aehnlichkeitsmass für gesprochene Einzelwörter

Studienarbeit SA-2004-13
Wintersemester 2003/2004

Betreuer: Rene Beutler

Verantwortlicher:
Prof. Dr. Lothar Thiele

6.2.2004

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

INHALTSVERZEICHNIS

Vorwort

Diese Arbeit richtet sich an Leser, die sich für die Spracherkennung interessieren. Es wird kein spezifisches Wissen benötigt um den folgenden Text verstehen zu können. Die verwendeten Algorithmen wurden in Matlab implementiert. Kenntnisse in diesem Programm sind sicher hilfreich aber nicht notwendig. Ich hoffe, dass ich interessierten Lesern mit dieser Arbeit Appetit auf mehr machen kann, so dass diese Arbeit eventuell sogar weitergeführt wird.

1 EINLEITUNG

1.1 Ausgangslage/Problemstellung

Ob sich zwei gesprochene Wörter ähnlich sind, können wir intuitiv recht gut beantworten. Wir nutzen diese Fähigkeit zum Beispiel, um jemanden zu verstehen, der Deutsch nicht als Muttersprache spricht und deshalb ab und zu Fehler macht. Wenn jemand abends sagt: "Ich lege mich ins Beet (b e: t)", werden wir wohl annehmen, dass er das offene und geschlossene ‚e' verwechselt hat und eigentlich Bett (b E t) meinte, weil das sehr ähnlich klingt und zudem auch semantisch viel mehr Sinn macht.

Auch für maschinelle Sprachverarbeitung kann es sehr hilfreich sein, wenn man die Ähnlichkeit der Aussprache zweier Wörter bestimmen kann. Ein Grund liegt zum Beispiel darin, dass Spracherkenner sehr empfindlich sind gegenüber Eingabefehlern. Übergibt man dem Spracherkenner ein falsch geschriebenes Wort, welches er nicht im Vokabular hat, erkennt er es nicht. Wenn er aber nach ähnlichen Wörtern suchen kann, findet er vielleicht so das richtige Wort.

Während das Problem intuitiv sehr einfach zu lösen ist, ist ein streng mathematisches Modell schwieriger zu erfassen. Eine algorithmische Lösung muss insbesondere in der Lage sein, Wörter mit unterschiedlicher Lautanzahl zu verarbeiten, womit also nicht nur Ersetzungen von Lauten, sondern auch Auslassungen und Einfügungen berücksichtigt werden müssen. Abgesehen davon muss Zwecks Vergleichbarkeit das Mass unabhängig von der Länge der Wörter sein und einen vordefinierten Wertebereich, z.B. [0,1], haben.

1.2 Zielsetzung

In meiner Semesterarbeit werde ich zuerst auf die Theorie eingehen, wie zwei Wörter miteinander verglichen werden. Vorhandene Verfahren sollen implementiert werden und die Funktionsweise anhand von Testwörtern validiert werden. In einem weiteren Schritt versuche ich neue Distanzmasse zu entwerfen. Diese neuen Masse werden in weiteren Testserien untereinander verglichen. Dabei sollen Aussagen über die Funktionsweise gemacht werden.

1.3 Vorgehensweise

Zuerst werde ich nach Fachliteratur in Bibliotheken und im Internet suchen. Sobald ich das Konzept verstanden habe, werde ich den Algorithmus mit Hilfe von Matlab implementieren. Dafür werden die Distanzen von Covington verwendet. Funktioniert der Algorithmus, versuche ich zwei neue Ähnlichkeitsmasse zu entwerfen. Damit diese Masse miteinander verglichen werden können, werde ich eine Sammlung von Testwörtern zusammenstellen. Am Schluss vergleiche ich mit Hilfe dieser Wörter die drei Ähnlichkeitsmasse und versuche eine Aussage über ihr Verhalten zu machen.

2 THEORETISCHER HINTERGRUND

2.1 Laute, Phone

Die kleinsten Einheiten der Lautsprache sind die Laute oder Phone. In dem System, das hier verwendet wurde, existieren 38 Laute. Mit diesen Lauten können alle Wörter dargestellt werden. Im Anhang findet man eine Tabelle mit den Lauten, inklusive Beispiele.

2.2 Editiersequenzen

Um zwei Wörter vergleichen zu können, betrachtet man die Anzahl der Operationen, die man auf ein Wort anwenden muss, um dieses in ein anderes zu überführen. Die Operationen werden nacheinander angewendet; so werden Schritt für Schritt die Wörter überführt.

Es gibt vier mögliche Editieroperationen, die an jeder Stelle angewendet werden können:

- ein Laut wird ersetzt durch einen Anderen (**R**eplace)
- zwei Laute sind gleich (**M**atch)
- es wird ein Laut eingefügt (**I**nsert)
- es wird ein Laut gelöscht (**D**elete)

Das folgende Beispiel soll dieses Vorgehen erläutern:

Das Wort ‚stinken' wird in das Wort ‚braten' überführt:

b r a: **t** @ **n**

S **t** I n k @ **n**

Das ergibt folgende Editiersequenz: D, D, R, M, I, I, I, M, M

2.3 Alignment

Unter Alignment versteht man den Vergleich zweier Wörter, von denen das eine das Referenzwort darstellt und das andere Wort in dieses überführt werden soll.

2.4 Editierdistanz

Um die Qualität des Alignments in Zahlen zu fassen, benutzt man Distanzen oder Ähnlichkeiten. Damit lässt sich dann auch das beste Alignment finden. Um diese Distanzen oder Ähnlichkeiten zu berechnen, führt man fiktive Kosten ein, die dann minimiert werden sollen. Für die Definition der Kosten für das Überführen von einzelnen Lauten benötigt man eine Kostenfunktion d(a,b), da die Wahrscheinlichkeit für das Ersetzen von verschiedenen Lauten unterschiedlich ist.

Es gibt drei Arten von Kostenfunktionen:

- d(a,b): Kosten für das Ersetzen von a durch b
- d(a,-): Kosten für das Löschen von a
- d(-,b): Kosten für das Einfügen von b

> **Definition 1:** Die Editierdistanz zweier Wörter ist die Summe der Kosten für die Editieroperationen, um das eine Wort in das andere zu überführen. Da es mehrere mögliche Editiersequenzen gibt, ist mit Distanz die kürzeste[1] Distanz gemeint.

Um ein optimales Alignment zu bekommen, müssen die Kosten, die entstehen, um ein Wort zu überführen, minimiert werden. Dazu bedient man sich der rekursiven Vorgehensweise der dynamischen Programmierung.

2.5 Dynamische Programmierung

Die dynamische Programmierung ist ein Verfahren, das in vielen Gebieten der Informatik angewendet wird. Die Grundidee ist das Problem(hier die minimale Editierdistanz) auf ein Teilproblem zurückzuführen, das einfach gelöst werden kann.

Die Bestandteile der dynamischen Programmierung sind Rekursion, Matrixerstellung und Traceback.

2.5.1 Rekursion

Zunächst wird das Problem rekursiv definiert und in kleinere Stücke zerteilt. Man berechnet die Distanz $D_{i,j}$ für $1 \le i \le n$ und $1 \le j \le m$, um schliesslich auf die Gesamtdistanz $D_{n,m}$ schliessen zu können. Jeder Rekursionsschritt baut auf den davor berechneten Schritten auf.

> **Definition 2:** Mit zwei Wörtern $a = a_1 a_2 ... a_n$ und $b = b_1 b_2 ... b_m$ definiere man die Distanzmatrix
>
> $$D_{i,j} = D(a_1 a_2 ... a_i, b_1 b_2 ... b_j)$$

Dies ist die rekursive Definition der Distanz-Matrix. Ein Eintrag (i,j) in der Matrix entspricht der Distanz der Teilwörter $a_1 ... a_i$ und $b_1 ... b_j$. Man betrachtet also zuerst Teilwörter und zum Schluss erst das komplette Alignment $D_{n,m}$.

> **Definition 3:** Der Rekursionsanfang für die Berechnung der Distanzmatrix beträgt

[1] Die kürzeste Distanz muss nicht immer die minimale Anzahl von Operationen bedeuten.

$$D_{0,0} = 0, D_{0,j} = \sum_{k=1}^{j} d(-, b_k), D_{i,0} = \sum_{k=1}^{i} d(a_k, -)$$

Satz 1: Geht man von einem optimalen vorangegangenen Schritt aus, berechnet sich der nachfolgende Schritt folgendermassen

$$D_{i,j} = \min \begin{Bmatrix} D_{i-1,j} + d(a_i, -), \\ D_{i-1,j-1} + d(a_i, b_j), \\ D_{i,j-1} + d(-, b_j) \end{Bmatrix}$$

2.5.2 Matrixerstellung

Zunächst wird die Matrix gemäss Definition 3 initialisiert. Daraus resultiert, dass die erste Zeile und die erste Spalte der Matrix mit Werten gefüllt werden. Der Rest der Matrix wird dann anhand von Satz 1 aufgefüllt. Bei jedem Eintrag der Matrix muss nur der Eintrag links, oben und links-oben betrachtet werden:

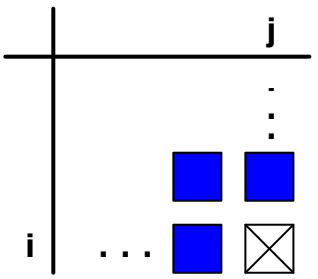

Abb. 1

2.5.3 Traceback

Man merkt sich bei jeder Position (i,j) einen oder mehrere Zeiger, der speichert, welche der drei Möglichkeiten (i-1,j),(i-1,j-1) oder (i,j-1) zu einer optimalen Lösung von $D_{i,j}$ geführt hat. Beginnend mit dem Element $D_{n,m}$ verfolgt man die Zeiger zurück und erhält so einen optimalen Pfad. Sind bei einer Position mehrere Zeiger vermerkt, gab es also mehrere Möglichkeiten zu diesem Distanzwert zu kommen, dann bedeutet das, dass es mehrere mögliche Pfade gibt.

2.6 Normierung

Für jedes Ähnlichkeitsmass wird die Editierdistanz normiert, so dass der normierte Wert im Bereich zwischen 0 und 1 liegt. Der Wert 0 soll bedeuten, dass zwei Wörter völlig unterschiedlich sind, der Wert 1, dass zwei Wörter identisch sind.

Die Normierungsgrenzen hierfür werden für jedes Ähnlichkeitsmass separat berechnet: Es wird immer ein minimaler Wert ‚minVal' bestimmt, welcher die Kosten für den bestmöglichen Fall aufweist (best case). Das entspricht dem Fall, wenn die beiden Wörter identisch sind.

Für die Normierung wird auch ein maximaler Wert ‚maxVal' benötigt, welcher die Kosten für den schlechtesten Fall beinhaltet (worst case). Der schlechteste Fall tritt ein, wenn die Wörter von Grund auf verschieden sind. Also theoretisch wenn jeder Laut auf den Laut abgebildet wird, der die höchsten Kosten verursacht.

Definition 4: Die Normierung der Editierdistanz berechnet sich wie folgt:

$$c = 1 - \frac{e - \min Val}{\max Val - \min Val}$$

Wobei mit e die Editierdistanz gemeint ist.

Die Abbildung 2 zeigt eine graphische Darstellung der Normierung:

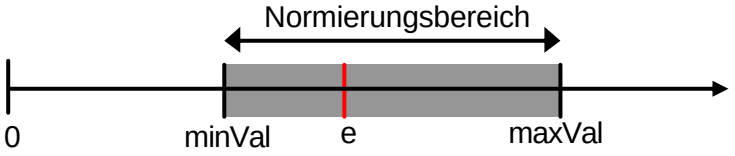

Abb. 2: Die Normierungsgrenzen

3 IMPLEMENTIERUNG

3.1 Funktionen

Die Funktionen habe ich in Matlab implementiert, da sich diese Anwendung gut eignet um Matrizen zu bearbeiten und graphische Outputs zu generieren. In Abbildung 3 ist der Aufbau der verwendeten Struktur dargestellt.

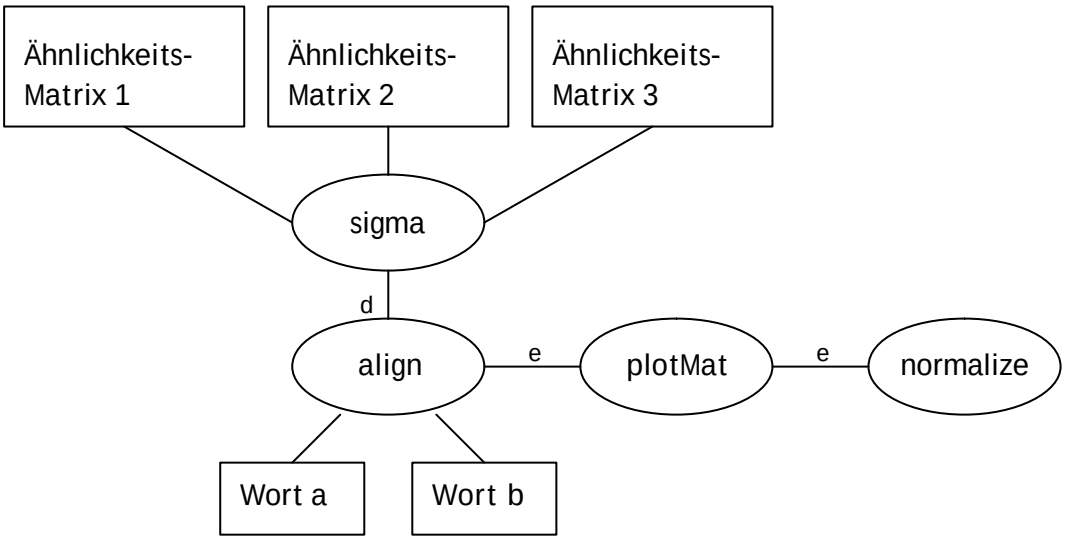

Abb. 3: Struktur der Funktionen

3.1.1 sigma(u,v,A,m,t)

sigma liest aus den Matrizen der Ähnlichkeitsmassen die gewünschten Distanzen heraus.

Als Parameter benötigt man die beiden Indizes der abzubildenden Laute, sowie die Matrix des gewünschten Ähnlichkeitsmasses. Weiter braucht man zwei Hilfsparameter m und t, die den Modus und die Editieroperation festlegen.

Als Ausgabewert erhält man die Distanz d.

Funktionsablauf:

Je nach Modus und Editieroperation wird anhand der Indizes in der Matrix der richtige Wert selektiert und zurückgegeben.

3.1.2 align(a,b,A,m)

Die zentrale Funktion bildet align. Sie führt die ganze dynamische Programmierung durch und erzeugt die graphische Darstellung.

Als Parameter benötigt sie die beiden Wörter, die miteinander verglichen werden sollen, wobei das erste Wort das Referenzwort darstellt. Weiter braucht sie eine Ähnlichkeitsmatrix und den, aus der gewählten Matrix resultierenden, Modus.

Als Ausgabewert gibt sie die berechnete Editierdistanz e zurück.

Funktionsablauf:

Zuerst werden die Laute der beiden Wörter anhand einer Referenztabelle in Indizes umgewandelt. Falls ein Wort einen falschen Laut enthält, wird eine Fehlermeldung ausgegeben.

Mit der Funktion ‚sigma' werden die benötigten Distanzen mittels Indizes aus der Matrix ausgelesen. Es werden zwei Fälle unterschieden: Im Modus 1 werden Löschen und Einfügen als Ersetzungen modelliert. Bei der benötigten Ähnlichkeitsmatrix handelt es sich also um eine 38*38 Matrix. Im Modus 0 werden diese beiden Editieroperationen von einer Ersetzung unterschieden. Als Matrix wird hier eine 39*39 Matrix verwendet. Die zusätzliche Zeile und Spalte der Matrix beinhaltet die Distanzen für das Löschen und Einfügen der Laute.

Basierend auf der dynamischen Programmierung wird die Matrix D gebildet. Um den gewählten Weg zu speichern wird eine Hilfsmatrix E erzeugt, die, je nachdem, welche der drei Möglichkeiten (i-1,j),(i-1,j-1) oder (i,j-1) gewählt wurde, eine 1, eine 3 oder eine 2 speichert.

Am Schluss wird der Graph geplottet. Die Zusammensetzung des Graphen wird später erläutert.

3.1.3 plotMat(a,b,A,B,C)

plotMat erzeugt für jede der drei Matrizen einen ‚subplot', so dass man die Resultate der drei Ähnlichkeitsmasse, da sie in demselben Graph dargestellt sind, gut vergleichen kann.

Als Parameter benötigt plotMat die zu vergleichenden Wörter und die drei Matrizen der Ähnlichkeitsmasse.

Als Ausgabewert wird ein Vektor der Länge 3 zurückgegeben, welcher die für jedes Ähnlichkeitsmass berechnete Editierdistanz enthält.

Funktionsablauf:

Für jeden ‚subplot' wird die Funktion ‚align' mit den benötigten Parametern aufgerufen und der so erhaltene Wert in den Vektor e gespeichert.

3.1.4 normalize(a,b,A,B,C)

Diese Funktion normiert die drei Werte von e, so dass sie zwischen 0 und 1 zu liegen kommen.

Als Parameter werden wiederum die beiden Wörter und die drei Matrizen benötigt.

Als Ausgabewert erhält man einen Vektor der Länge 3 mit den drei normierten Werten.

Funktionsablauf:

Zuerst wird überprüft, welches der beiden Wörter länger ist. Anhand dieses Vergleichs werden zwei Fallunterscheidungen bei der Berechnung der Normierungsgrenzen gemacht.

Danach wird die Funktion plotMat ausgeführt. Dadurch erhält man die benötigten Editierdistanzen, die normiert werden müssen.

Die Bestimmung der Normierungsgrenzen erfolgt, wie auch die Normierung selbst, für jedes Ähnlichkeitsmass separat. Die Idee ist, dass anhand der Matrizen der Ähnlichkeitsmasse und der Länge des längsten Wortes die Minimal- und Maximalwerte berechnet werden. Man sucht also in den für die beiden Wörter relevanten Einträgen der Matrizen nach dem Minimum und dem Maximum und multipliziert diese mit der Länge des längsten Wortes. Am Schluss werden die Werte von e über diesen Bereich normiert.

Die Resultate der Normierung werden im Vektor c gespeichert.

3.2 Ähnlichkeitsmasse

Zuerst wird das bestehende Mass von Covington implementiert und danach zwei neue Masse entworfen. Das Implementieren der Ähnlichkeitsmasse beinhaltet das Definieren der Distanzen für alle Laute und das Generieren der daraus folgenden Matrix.

3.2.1 Covington (1996)

Das Ähnlichkeitsmass von Covington sieht man in Abbildung 4. Das Mass ist sehr einfach; es benutzt keine lautspezifischen Eigenschaften und unterscheidet nur drei verschiedene Typen: Konsonanten, Vokale und „glides“. Mit „glides“ sind die beiden Laute ‚w’ und ‚y’ gemeint. Viele wichtige Charakteristiken von Lauten, wie Ort oder Art der Artikulation, werden ignoriert. Zum Beispiel werden die Wörter ‚Nase’ und ‚Tabu’ beide einfach als „Konsonant-Vokal-Konsonant-Vokal“-Folge behandelt. Diese schwache „Auflösung“ ist definitiv ein Nachteil dieses Masses. Ein weiterer Schwachpunkt ist, dass dieses Mass auf keiner theoretischen Basis beruht, sondern durch „trial and error“ entwickelt wurde. Covington erkannte selber, dass sein Mass nur als Grundlage für ein weiteres, genaueres Ähnlichkeitsmass verwendet werden kann.

Dieses Mass verwende ich als erstes Ähnlichkeitsmass. Allerdings in abgeänderter Form: In meiner Umsetzung des Modells von Covington unterscheide ich nur zwischen Konsonanten und Vokalen. Die „glides“ sind zwischen den Konsonanten und den Vokalen aufgeteilt: das

‚w' ist ein Konsonant und das ‚y' ein Vokal. Weiter habe ich, im Gegensatz zu Covington, nicht unterschieden, ob einem Löschen oder Einfügen schon die gleiche Editierfunktion vorangegangen war. Die dritte Vereinfachung ist, dass die Kosten für eine Abbildung eines ‚i' und eines ‚y' oder eines ‚u' und eines ‚w' nicht separat betrachtet werden. Das genaue Ähnlichkeitsmass, so wie ich es verwende, ist in Abbildung 5 dargestellt.

Kosten	Bedingungen
0	Konsonanten oder „glides" sind identisch
5	Vokale sind identisch
10	Vokale unterscheiden sich nur in der Länge, oder Abbildung von ‚i' und ‚y', oder ‚u' und ‚w'
30	Vokale sind unterschiedlich
60	Konsonanten sind unterschiedlich
100	Vokal wird auf Konsonant abgebildet oder Konsonant auf Vokal
50	Löschen eines Lautes oder Einfügen eines Lautes
40	Löschen oder Einfügen eines Lautes, wenn schon ein Löschen oder Einfügen vorangegangen war

Abb. 4: Ähnlichkeitsmass von Covington (1996)

Kosten	Bedingungen
0	Konsonanten sind identisch
5	Vokale sind identisch
10	Vokale unterscheiden sich nur in der Länge
30	Vokale sind unterschiedlich
60	Konsonanten sind unterschiedlich
100	Vokal wird auf Konsonant abgebildet oder Konsonant auf Vokal
50	Löschen eines Lautes oder Einfügen eines Lautes

Abb. 5: Ähnlichkeitsmass von Covington (abgeändert)

Die Funktion Covington()

Die Implementierung der Matrix für Covington ist ganz einfach: Zuerst werden vier Arrays erzeugt, die definieren, welche Laute Vokale sind, welche Laute Konsonanten sind und wenn ein Laut ein Vokal ist, ob es noch einen zweiten Laut gibt, der sich nur in der Länge von ihm unterscheidet. Aufgrund dieses Wissens kann die Matrix jetzt aufgebaut werden. Das Resultat ist eine 39*39 Matrix, welche in der 39. Zeile und der 39. Spalte die Kosten für das Einfügen, bzw. das Löschen beinhaltet. In den Zeilen und Spalten mit den Indizes 1 bis 38 stehen die Kosten für das Ersetzen eines Lautes durch einen anderen. In Abbildung 6 ist ein Ausschnitt aus der Covington Matrix abgebildet.

	a	a:	a_I	a_U	b	C
a	5	10	30	30	100	100
a:	10	5	30	30	100	100
a_I	30	30	5	30	100	100
a_U	30	30	30	5	100	100
b	100	100	100	100	0	60
C	100	100	100	100	60	0

Abb. 6: Die Covington Matrix

Bei der Covington Matrix handelt es sich um eine symmetrische Matrix.

3.2.2 Confusion Matrix

Als zweites Ähnlichkeitsmass habe ich die Confusion Matrix entworfen. Die Idee, welche hinter diesem Mass steckt, ist, dass man die Wahrscheinlichkeits-Werte für das Erkennen von einzelnen Lauten von einem reellen Spracherkenner als Distanzen verwendet.

Das funktioniert folgendermassen: Wenn man einen Laut einem Spracherkenner übergibt, liefert dieser eine Wahrscheinlichkeits-Tabelle mit einem Wert für jeden Laut, den er in seinem Vokabular hat. Dieser Wahrscheinlichkeits-Wert gibt an, mit welcher Wahrscheinlichkeit dieser Laut erkannt wurde. Um dies zu verdeutlichen ein Beispiel:

> Dem Spracherkenner wird der Laut ‚b' übergeben. Als Resultat erhält man die Wahrscheinlichkeit mit der ein Laut erkannt wurde. Abbildung 7 zeigt einen Ausschnitt aus den berechneten Wahrscheinlichkeiten.

a	a:	a_I	a_U	b	C	d	e:	E	E:
0.009	0.009	0.002	0.011	0.169	0.002	0.044	0.003	0.005	0.001

Abb. 7: Wahrscheinlichkeiten für das Erkennen der Laute

Die Confusion Matrix enthält nun aber nicht genau diese Werte, da ja hier ein grosser Wahrscheinlichkeits-Wert bedeutet, dass zwei Laute identisch sind (also richtig erkannt wurden), die „align"-Funktion aber den optimalen Weg sucht, indem sie die geringsten Kosten wählt. Darum wird in die Confusion Matrix jeweils „1 minus die Wahrscheinlichkeit" eingefügt, so dass ein grosser Wert auch grosse Kosten bedeutet.

Die Funktion confMat()

Das Generieren dieser Matrix funktioniert folgendermassen: Zuerst wird die Datei „confusionmatrix.txt" geladen. Diese Datei enthält die Wahrscheinlichkeiten aller Laute, die der Spracherkenner im Vokabular hat. Aus dieser Datei werden dann die benötigten Wahrscheinlichkeiten herausgelesen und an den richtigen Stellen in der Matrix eingesetzt. Als Resultat erhält man eine 38*38 Matrix, die für jeden Laut einen Eintrag hat. In Abbildung 8 sieht man einen Ausschnitt der generierten Matrix.

	a	a:	a_I	a_U	b	C
a	0.709	0.753	0.945	0.87	0.991	1
a:	0.736	0.657	0.946	0.856	0.992	1
a_I	0.939	0.942	0.591	0.977	0.997	0.996
a_U	0.872	0.874	0.973	0.688	0.993	1
b	0.991	0.991	0.998	0.989	0.831	0.998
C	1	1	0.997	1	0.996	0.536

Abb. 8: Die Confusion Matrix

Bei der Confusion Matrix handelt es sich um eine **nicht** symmetrische Matrix.

3.2.3 Feature Matrix

Als drittes und letztes Ähnlichkeitsmass habe ich die Feature-Matrix entworfen. Die Idee, die diesem Mass zu Grunde liegt, ist die Tatsache, dass jeder Laut eine bestimmte Anzahl an Eigenschaften, so genannte Features, hat. Ich möchte nun zuerst diese Features ein bisschen genauer betrachten und dann mit der Implementierung fortfahren.

Wie schon erwähnt, verwendet unser System 38 Laute. Diese werden in Konsonanten und Vokale unterteilt. Insgesamt gibt es 27 Features, wobei zwei davon definieren, ob der Laut ein Konsonant oder ein Vokal ist.

Vokale zeichnen sich vor allem dadurch aus, dass sie stimmhaft sind. Anhand der Position des Zungenrückens können sie weiter unterteilt werden. Ist der Zungenrücken vorne im Mundraum, so spricht man von hellen Vokalen(i:,e:,E,...), ist er hingegen hinten, von dunklen Vokalen(U,u:,O,...). Der Zungenrücken liegt bei geschlossenen Vokalen hoch(U,i:,u:) und bei offenen Vokalen tief(a,a:). Es gibt noch zwei weitere Einteilungskriterien für Vokale: die Muskelspannung und die Lippenform. Auf diese beiden Kriterien möchte ich aber nicht genauer eingehen. Der Abbildung 9 kann man die Features für alle Vokale entnehmen.

		Zungenstellung						
		vorn				neutral	hinten	
Muskelspannung		ge-spannt	unge-spannt	ge-spannt	unge-spannt	ungespannt	ge-spannt	unge-spannt
Zungen-höhe	hoch	i:	I	y:	Y		u:	U
	mittel	e:		ox:	oe	@	o:	O
	gehoben	E:	E					
	tief					a: a		
		ungerundet		ungerundet		ungerundet	ungerundet	
		Lippenform						

Abb. 9: Features für die Vokale

Neben den Vokalen, die aus einem Laut bestehen, gibt es auch noch die so genannten Gleitlaute (Diphthonge), die aus zwei Vokalen zusammengesetzt sind. In unserem System gibt es drei Diphthonge: ‚a_I', ‚a_U' und ‚O_Y'. Zur Vereinfachung habe ich bei den Diphthongen die Vereinigung der Features der einzelnen Laute genommen. Für den Diphthong ‚a_I' wäre das zum Beispiel die Vereinigung der Features van ‚a' mit denjenigen von ‚I'.

Die Konsonanten können nach Artikulationsort und –art differenziert werden. Ich möchte aber nicht näher auf diese Features eingehen. Für Interessierte empfehle ich die Vorlesung „Sprachverarbeitung I" und das zugehörige Skript (siehe Literaturverzeichnis). Die verschiedenen Features für die Konsonanten sieht man in der Abbildung 10.

		Artikulationsstelle/Artikulationsort						
		bi-labial	labio-dental	dental	alveo-lar	palatal	velar	glottal
explosiv	sth.	b			d		g	
	stl.	p			t		k	
frikativ	sth.		v	z		j		
	stl.		f	s	S	C	x	h
nasal		m			n		N	
lateral					l			
intermittierend					r			

Abb. 10: Features für die Konsonanten

Wenn man nun annimmt, dass ähnliche Laute auch ähnliche Features aufweisen, kann man über die Anzahl unterschiedlicher Features eine Distanz definieren.

Definition 5: Die Distanzen der Feature-Matrix berechnen sich wie folgt:

$$d(a_i, b_j) = \sum_{f=1}^{27} abs(\, feature_i(a) - feature_i(b))$$

$feature_i$ ist gleich 1 wenn der Laut das i-te Feature hat. Sonst ist es gleich 0.

Die Distanz zweier Laute berechnet sich also, indem man die Zahl der unterschiedlichen Features addiert.

Die Funktion featMat()

Zuerst wird definiert, welcher Laut zu welchen Features gehört. Dies geschieht indem für jedes Feature ein Array mit den dazugehörigen Lauten angelegt wird. Für jeden Laut wird dann jedes Feature überprüft, ob der Laut darin vorkommt und ob der zu vergleichende Laut ebenfalls darin vorkommt. Kommen beide darin vor oder keiner von beiden, dann passiert nichts. Hat aber nur der eine Laut das Feature und der andere nicht, wird die featSum, also die Anzahl unter-

schiedlicher Features, um eins erhöht. Das Resultat ist eine 38*38 Matrix, welche die Anzahl der unterschiedlichen Features für jedes Laut-Paar enthält. Ein kleiner Ausschnitt der Feature Matrix ist in Abbildung 11 dargestellt.

	a	a:	a_I	a_U	b	C
a	0	0	1	3	8	10
a:	0	0	1	3	8	10
a_I	1	1	0	4	9	11
a_U	3	3	4	0	11	13
b	8	8	9	11	0	6
C	10	10	11	13	6	0

Abb. 11: Die Feature Matrix

Bei der Feature Matrix handelt es sich um eine symmetrische Matrix.

3.3 Normierung

Ich möchte hier genauer auf die Berechnung der Minimal- und Maximalwerte der Normierung für die verschiedenen Ähnlichkeitsmasse eingehen.

3.3.1 Confusion Matrix

Für die Confusion Matrix wird der minVal so bestimmt, dass für jeden Laut des ersten Wortes, die Kosten für das Ersetzen des Lautes durch denselben Laut aus der Matrix ausgelesen wird und aus diesen Werten dann die Summe gebildet wird.

Um den maxVal zu berechnen, werden die maximalen Kosten für jeden Laut in der Confusion Matrix gesucht und mittels dieser Werte die Summe erstellt.

An dieser Stelle muss auf einen Spezialfall eingegangen werden: Falls das zweite Wort länger ist als das erste, würde die Normierung eine Verzerrung des Wahrscheinlichkeits-Werts bewirken. Das folgende Beispiel verdeutlicht diese Verzerrung:

> Verglichen werden die beiden Wörter: Verwalter und Walter. Im ersten Versuch ist das längere Wort ‚Verwalter' an der ersten Stelle (also das

erste Wort) und das Wort ‚Walter' an der zweiten Stelle (also das zweite Wort).

normalize('f E r v a l t @ r', 'v a l t @ r', A,B,C);

gibt den Wahrscheinlichkeits-Wert 0.63981 zurück.

Werden nun aber in einem zweiten Versuch die beiden Wörter vertauscht, so dass das längere Wort ‚Verwalter' als erstes Wort verwendet wird, kann man die Verzerrung des Wahrscheinlichkeits-Werts beobachten.

normalize('v a l t @ r', 'f E r v a l t @ r',A,B,C);

Gibt den Wahrscheinlichkeits-Wert -1.1029 zurück. Also einen unbrauchbaren Wert. Dieser Wert resultiert daraus, dass der minimale und der maximale Wert nicht mehr stimmen, da sie anhand des ersten Wortes berechnet wurden und darum viel zu klein sind.

Um dieses Fehlverhalten zu umgehen, habe ich eine Fallunterscheidung eingeführt, welche die Normierung anpasst, falls das zweite Wort länger ist als das erste. Diese Anpassung beinhaltet, dass der minimale Wert um einen Wert mmin*(length(word2)-length(word1)) erhöht wird. Der Wert mmin entspricht den kleinsten Kosten, die benötigt werden, um einen beliebigen Laut auf einen Laut des ersten Wortes abzubilden. Das Resultat, das in mmin gespeichert ist, gibt mir die minimalen Kosten mit denen ich zusätzlich rechnen muss, wenn das erste Wort einen Laut kürzer ist als das zweite Wort. Wenn ich nun mmin mit dem Längenunterschied der beiden Wörter multipliziere erhalte ich die gewünschten minimalen Kosten. Das gleiche Vorgehen wird beim maximalen Wert angewendet. Der einzige Unterschied besteht darin, dass ich nicht die kleinsten Kosten suche, sondern die Grössten. Für den maximalen Wert maxVal entspricht mmax den maximalen Kosten, wenn ich einen zusätzlichen Laut auf das Wort abbilden muss. Dieser Wert wird auch wieder mit dem Längenunterschied der beiden Wörter multipliziert und dann zum maximalen Wert maxVal addiert. Das folgende Beispiel soll dieses Vorgehen verdeutlichen:

Es werden wieder die gleichen beiden Wörter ‚Walter' und ‚Verwalter' miteinander verglichen. ‚Walter' ist das erste Wort und ‚Verwalter' das Zweite. Ohne die Fallunterscheidung ergibt die Normierung folgende Werte für minVal und maxVal: e(1) = 7.5120, minVal = 4.6100, maxVal = 5.9900. e(1) ist der Wert der durch das Alignment berechnet wurde.

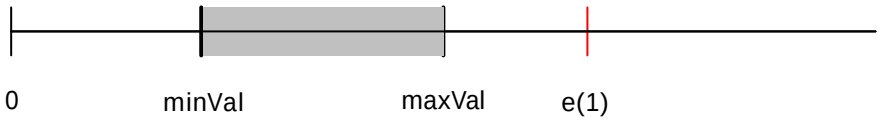

Abb. 12: Falsche Normierungs-Werte

In Abbildung 12 sieht man gut, wie die Normierungsgrenzen minVal und maxVal falsch gewählt wurden: Der zu normierende Wert e(1) liegt ausserhalb des Normierungsbereichs.

Mit der Fallunterscheidung resultieren folgende Werte für minVal und maxVal: e(1) = 7.5120, minVal = 6.9190, maxVal = 8.9900.

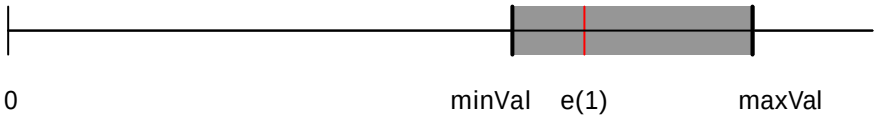

Abb. 13: Richtige Normierungs-Werte

Durch die Fallunterscheidung werden die Normierungsgrenzen nach oben verschoben. e(1) liegt jetzt im Normierungsbereich und es resultiert ein vernünftiger Wert. Der Wahrscheinlichkeits-Wert ist aufgrund dieser Normierung grösser als der Wahrscheinlichkeits-Wert, den man erhält, wenn man die beiden Wörter vertauscht.

3.3.2 Feature Matrix

Die Normierung bei der Feature Matrix verläuft analog zur Normierung bei der Confusion Matrix. Es wird auch hier wieder eine Fallunterscheidung gemacht, wenn das zweite Wort länger ist als das erste. Der einzige Unterschied zwischen den beiden Normierungen besteht in den gewählten Werten für minVal und maxVal. Die Berechnung ist dieselbe. Der minimale Wert bei einem Alignment zweier gleichlanger Wörter mit der Feature Matrix ist immer Null. Also minVal=0. Dies bedeutet, dass die beiden Wörter identisch sind. Der maximale Wert maxVal wird analog zur Normierung der Confusion Matrix berechnet: Es werden die höchsten Kosten für jeden Laut bestimmt und addiert.

Bei der Fallunterscheidung, wenn also das zweite Wort länger ist als das erste, verändert sich nur der maximale Wert maxVal und zwar analog zur Normierung bei der Confusion Matrix. minVal bleibt theoretisch 0, auch wenn dieser Wert in der Praxis kaum möglich ist. Ein Minimalwert von 0 wird ja erreicht, wenn alle Laute identisch sind. Wenn die beiden Wörter aber unterschiedlich lang sind, ist dies fast nicht möglich. Es gibt fast keine Wörter, welche dieselben Laute hintereinander enthalten. Ausnahmen sind zum Beispiel

doppeltes ‚t' oder doppeltes ‚n', usw. Im Normalfall ist also der Minimalwert grösser als 0. Das folgende Beispiel soll diesen Umstand besser erläutern:

> Wenn man zum Beispiel ein Wort sucht, welches in einem Vergleich mit dem Wort ‚das' den Wahrscheinlichkeits-Wert 0 ergibt und das länger ist als das Wort ‚das' findet man keine reellen Wörter. Es kommen nur Wörter in Frage, welche einen der drei Laute vom Wort ‚das' mehrmals hintereinander im Wort haben. Also zum Beispiel ‚ddddas' oder ‚daaas'. Diese Wörter können höchstens entstehen, wenn die sprechende Person stottert.

Es handelt sich also bei minVal um einen theoretisch erreichbaren Wert, welcher in der Praxis aber selten bis nie erreicht wird.

3.3.3 Covington Matrix

Die Normierung für Wörter gleicher Länge oder mit längerem ersten Wort erfolgt analog zu den anderen Verfahren.

Wenn das zweite Wort länger ist als das erste, wird bei der Covington Matrix-Normierung der minimale Wert nicht verändert. Der maximale Wert wird im Unterschied zu den anderen beiden Massen aber nicht mit der höchsten Strafe bemessen, sondern mit den Kosten für ein Einfügen eines Lautes. Dies ergibt einen passenderen Wert im Vergleich zu den anderen Ähnlichkeitsmassen, als wenn man maxVal mit den höchsten Kosten berechnet.

3.4 Der Graphische Output

Die Funktion ‚align' stellt die Matrix mit den Distanzen für die einzelnen Laute als Flächen mit unterschiedlichen Graustufen dar. Je dunkler dieser Grauton ist, desto kleiner ist die dazugehörige Distanz. Mittels der dynamischen Programmierung wurde der optimale Pfad berechnet. Dieser wird nun mit einer roten Linie dargestellt. Diese Linie sollte so häufig wie möglich durch dunkle Bereiche verlaufen, da diese ja die günstigsten Distanzen darstellen. Die drei Distanzmatrizen können unterschiedliche optimale Pfade für die gleichen zwei Wörter generieren, darum muss die rote Linie also nicht immer identisch verlaufen. Die Abbildung 22 zeigt den graphischen Output für den Vergleich der beiden Wörter ‚Tausender' und ‚Ende'.

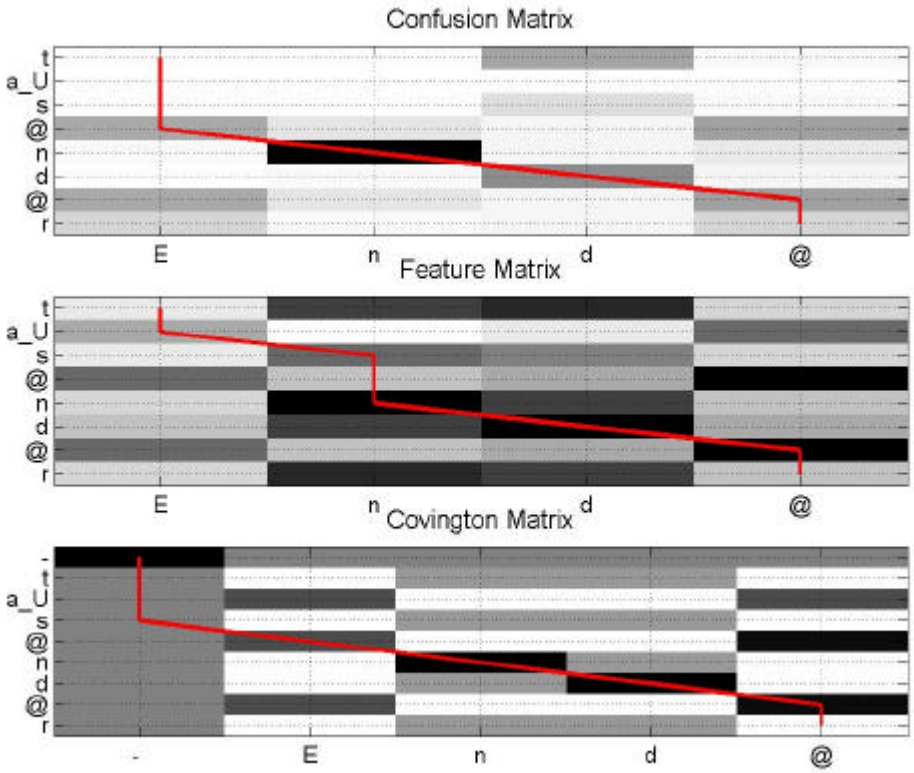

Abb. 14: Der graphische Output

In dieser Abbildung sieht man, dass die Feature Matrix einen anderen optimalen Pfad bestimmt hat als die Covington und die Confusion Matrix. Bei der Confusion Matrix sieht man auch schön, wie die rote Linie in den dunklen Flächen verläuft.

3.5 Das Ausführen eines Alignments

Um einen Vergleich zweier Wörter durchzuführen, müssen zuerst die drei Matrizen initialisiert werden. Dies geschieht mit den folgenden Befehlen:

```
A=confMat;
B=featMat;
C=covington;
```

Danach müssen die beiden Wörter deklariert werden: z.B.

```
a='t a_U s @ n d @ r';
b='E n d @';
```

Mittels des Aufrufs der Normalize-Funktion wird das Alignment ausgeführt:

```
c=normalize(a,b,A,B,C);
```

Als Ergebnis werden die normierten Werte in den Vektor c gespeichert und die graphische Darstellung wird erzeugt.

4 VERSUCHE

Um die Ähnlichkeitsmasse zu testen, habe ich ein Set von Testwörtern zusammengestellt. Diese sind im Anhang aufgeführt.

Im Folgenden habe ich diverse Versuche durchgeführt, um die drei Distanzmasse untereinander zu vergleichen und so Gemeinsamkeiten und Unterschiede feststellen zu können. Die Resultate sind aber mit Vorsicht zu geniessen. Die tatsächlichen Werte hängen sehr von den gewählten Lauten ab und auch die Anzahl der gemachten Versuche, aufgrund der fehlenden Zeit für diese Arbeit, ist zu klein, um eine sichere Aussage zu machen.

Bei allen Versuchen wird immer das erstgenannte Wort als Referenzwort verwendet. Für alle Versuche gilt die Legende von Abbildung 15.

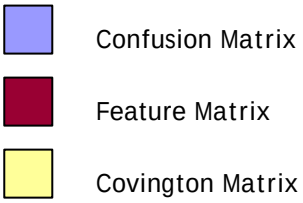

Abb. 15: Legende zu den Diagrammen

4.1 Wörter gleicher Länge

In einer ersten Versuchsanordnung werden Wörter gleicher Länger miteinander verglichen und dabei die unterschiedlichen Verhaltensweisen der Matrizen untersucht. Identische Wörter ergeben bei allen drei Matrizen einen Wahrscheinlichkeits-Wert von 1. Unterschieden werden die beiden Fälle:

- der unterschiedliche Laut ist ein Konsonant
- der unterschiedliche Laut ist ein Vokal

4.1.1 Der unterschiedliche Laut ist ein Konsonant

Verglichen werden im ersten Word-Set die drei Wörter Haus, Maus und Laus. Im zweiten Word-Set die drei Wörter Falter, Walter und Halter. In einem letzten Word-Set vergleiche ich die Wörter Tausender, lausender und mausender. Ziel dieses Versuchs ist zu verifizieren, dass die Ähnlichkeitsmasse richtig funktionieren. Ich erwarte von allen drei Massen Werte zwischen 0.8 und 1.

Versuch:

Word-Set 1: set={‘h a_U s’,’m a_U s’,’l a_U s’}

Word-Set 2: set={‘f a l t @ r’,’v a l t @ r’,’h a l t @ r’}

Word-Set 3: set={‘t a_U s @ n d @ r’,’l a_U s @ n d @ r’,’m a_U s @ n d @ r’}

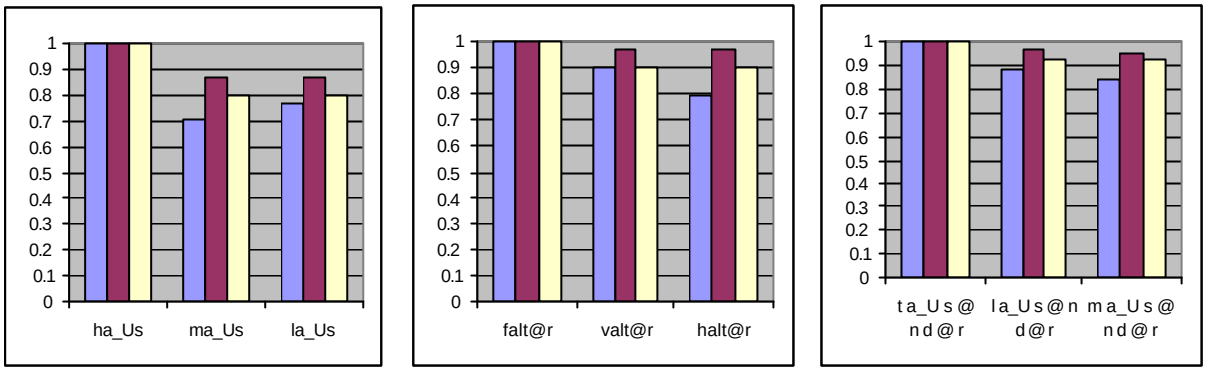

Abb. 16: WSK-Werte von Word-Set 1, Word-Set 2 und Word-Set 3

Resultate:

Die Resultate schauen vernünftig aus. Im Word-Set 1 sind die Werte zwischen 0.7 und 0.9, im Word-Set 2 sind sie zwischen 0.8 und 1 und im Word-Set 3 zwischen 8.3 und 1. Da es sich beim ersten Word-Set um sehr kurze Wörter handelt, fällt ein unterschiedlicher Laut mehr ins Gewicht als bei längeren Wörtern. Mit der Covington Matrix und der Feature Matrix erhalten wir Werte, wie ich sie erwartet habe. Für die Confusion Matrix fallen die Werte eher zu klein aus. Theoretisch macht die Beobachtung, dass mit zunehmender Länge der Wörter auch die WSK steigt, auch Sinn, weil die Editierdistanz gleich bleibt (es ist immer nur ein Laut unterschiedlich). Das einzige was sich ändert, sind die Normierungsgrenzen. In der Folge steigt der maximale Wert und bewirkt eine Verschiebung von e in Richtung minimaler Wert und folglich eine Erhöhung der WSK.

Auch praktisch macht diese Beobachtung Sinn: Menschen können bei langen Wörtern schlechter unterscheiden, welches Wort jetzt gesagt wurde, wenn sich nur ein Laut unterscheidet. Bei kürzeren Wörtern würden sie schneller merken, wenn sich ein Laut unterscheidet.

4.1.2 Der unterschiedliche Laut ist ein Vokal

Es werden zwei Word-Sets gebildet, mit Wörtern gleicher Länge pro Word-Set. Im ersten Word-Set werden die Wörter denn, dann und den verglichen. Im zweiten Word-Set die Wörter Laken, Locken, Lücken und lecken. Ziel dieses Versuchs ist erneut die Prüfung ob die Werte im erwarteten Bereich liegen(zwischen 0.8 und 1).

Versuch:

Word-Set 4: set={'d E n','d a n','d e: n'}

Word-Set 5: set={'l a: k @ n','l O k @ n','l Y k @ n','l E k @ n'}

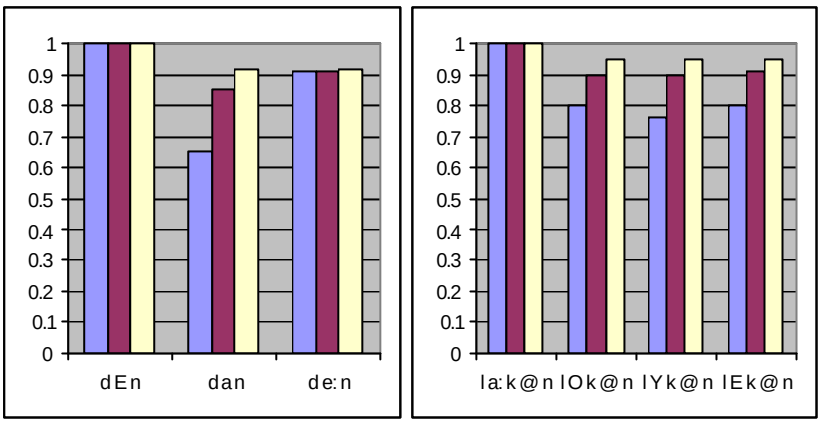

Abb. 17: WSK-Werte von Word-Set 4 und Word-Set 5

Resultate:

Man erhält wieder vernünftige Resultate. Die Werte sind wie schon beim vorherigen Versuch für die Confusion Matrix zu klein. Bei den kurzen Wörtern ist das noch stärker spürbar als bei den langen Wörtern. Covington und Feature Matrix liefern Werte im vermuteten Bereich. Was man aber noch anmerken muss, ist dass die Covington Matrix innerhalb eines Word-Sets immer die gleichen Werte liefert und darum nicht sehr aussagekräftig ist. Die Feature Matrix scheint für diesen Vergleich die besten Werte zu liefern, da sie innerhalb des gleichen Word-Sets noch weiter differenziert.

4.1.3 Völlig unterschiedliche Wörter

In diesem Versuch möchte ich untersuchen, wie sich die Werte verhalten, wenn man völlig unterschiedliche Wörter miteinander vergleicht. Ich erwarte, dass sich die Werte deutlich von den Werten unterscheiden, welche mit ähnlichen Wörtern erreicht wurden.

Versuch:

Word-Set 6: set={'h a t','E C o:','b a: n','h o: f','d e: n'}

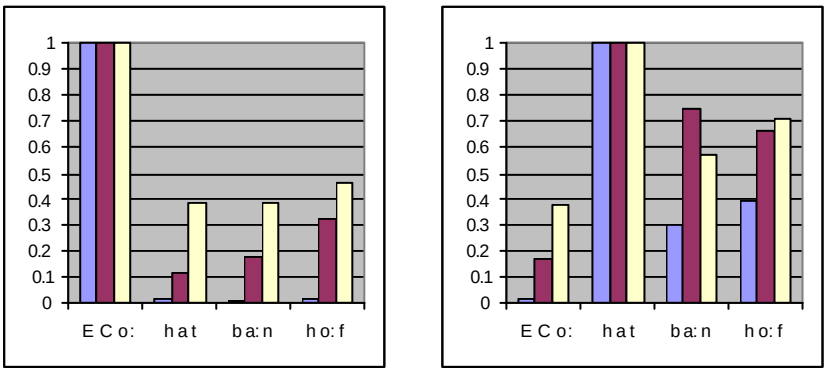

Abb. 18: WSK-Werte von Word-Set 6 Referenzwort ‚Echo' und Referenzwort ‚hat'

Resultate:

Dem linken Diagramm aus der Abbildung 18 kann man entnehmen, dass für das Referenzwort ‚Echo' die Werte von der Confusion Matrix sehr klein sind. Man kann fast keinen Unterschied erkennen zwischen den Werten der einzelnen Wörtern. Die Feature Matrix scheint die Unterschiede der einzelnen Wörter am besten zu erkennen. Die Werte liegen für alle drei Matrizen in einem vernünftigen Bereich, wobei sich die Covington Matrix eher an der oberen Grenze bewegt. Vor allem die Werte der Feature Matrix hätte ich tiefer erwartet und wenn man die Ergebnisse der Confusion Matrix mit dem linken Diagramm vergleicht, erscheinen auch diese Resultate zu hoch.

Da es sich um sehr kurze Wörter handelt, führe ich noch einen Versuch mit längeren Wörtern durch. Ich erwarte, dass die Werte deutlicher ausfallen.

Versuch:

Word-Set 7: set={'b @ h E n d @','k E t s C @ n','b a_U m S t a m','t O m b O l a'}

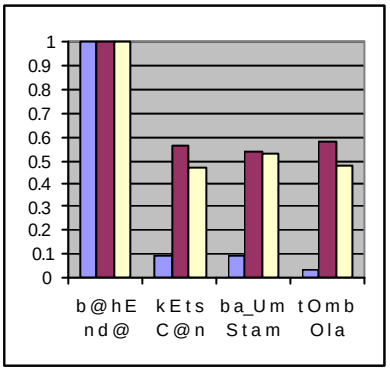

Abb. 19: WSK-Werte von Word-Set 7

Resultate:

Entgegen den Erwartungen sind die Ergebnisse nicht im gewünschten Bereich. Man kann erkennen, dass die Werte für die Feature Matrix und die Covington Matrix generell zu hoch sind für unterschiedliche Wörter. Dies resultiert daraus, dass die Normierung theoretisch gemacht wurde und sich nicht an reellen Wörtern orientiert. Das bedeutet, dass der maximale Wert für die Normierungsgrenze zu hoch gewählt wurde, damit auch ein theoretisch mögliches Wort (z.B. nur Vokale oder nur Konsonanten) im Normierungsbereich zu liegen käme. Praktisch würde man solche Wörter ausschliessen und könnte so die Normierungsgrenzen optimieren. Man bekäme so für unterschiedliche Wörter bessere, also kleinere, Werte. Ein zweiter Grund ist, dass je länger die Wörter sind, die dynamische Programmierung mehr Möglichkeiten hat, die beiden Wörter aufeinander abzubilden und darum ein besserer Wert entsteht, als zu erwarten wäre.

Für die Confusion Matrix erhalten wir sehr gute Werte, welche in einem realistischen Bereich liegen.

Man muss aber auch festhalten, dass die Werte deutlich kleiner sind, als die Werte von ähnlichen Wörtern.

4.2 Wörter unterschiedlicher Länge

In einer zweiten Versuchsanordnung werden Wörter unterschiedlicher Länge miteinander verglichen.

4.2.1 Wörter unterscheiden sich durch einen Laut

Die Wörter sind bis auf einen Laut identisch. Aufgrund der Seltenheit von Wörtern, die sich nur durch einen Vokal unterscheiden, werden nur Wörter mit einem unterschiedlichen Konsonanten betrachtet.

Im ersten Testsatz vergleiche ich die Wörter und, Hund, wund, und Bund. Im zweiten Testsatz verwende ich Alter, altern, Walter und Falter.

Wie bei gleich langen Wörtern erwarte ich von den Resultaten der WSK, dass sie zwischen 0.7 und 1 sind.

Versuch:

Word-Set 8: set={'U n t','h U n t','v U n t','b U n t'}

Word-Set 9: set={'a l t @ r','a l t @ r n','v a l t @ r','f a l t @ r'}

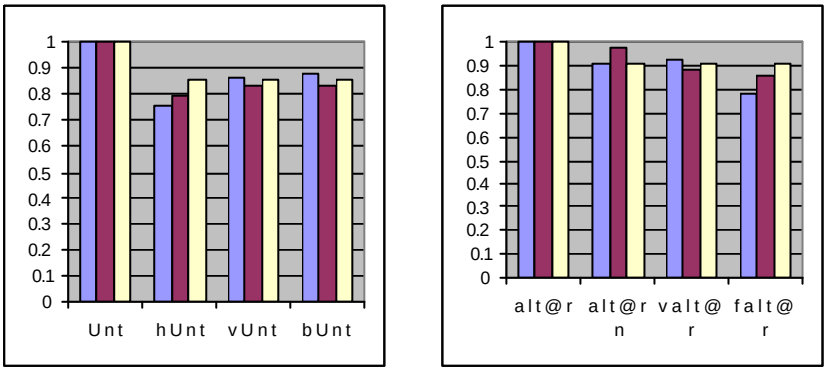

Abb. 20: WSK-Werte von Word-Set 8 und Word-Set 9

Resultate:

Die Werte liegen für alle drei Ähnlichkeitsmasse im Erwartungsbereich. Beiden Diagrammen kann entnommen werden, dass die Feature und die Confusion Matrix zum Teil grosse Unterschiede in der Grössenordnung von 0.1 aufweisen, auch wenn immer nur ein Laut geändert wurde. Im Gegensatz dazu steht das Covington Mass, welches überhaupt keine Differenzierung aufweist.

4.2.2 Wörter sind völlig unterschiedlich

In diesem Versuch soll wieder gezeigt werden, dass die Werte für unterschiedliche Wörter deutlich kleiner sind als die Werte für ähnliche Wörter.

Versuch:

Word-Set 10: set={'b a_U m a r k t','t O m b O l a','k a f e: m a S i: n @','a: l','E C o:','f O_Y @ r t s O_Y k','f E r v a l t @ r','h o: x h a_U s','r y: b @'}

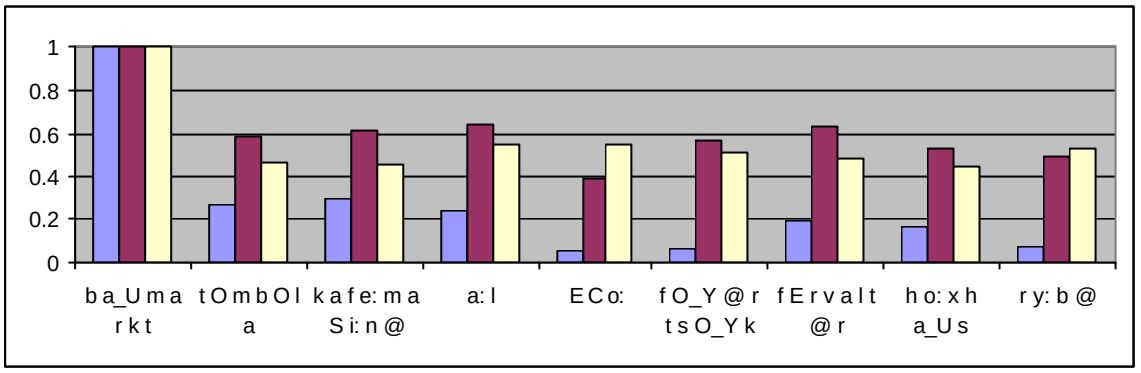

Abb. 21: WSK-Werte von Word-Set 10

Resultate:

Die Werte sind für alle drei Ähnlichkeitsmasse deutlich kleiner als für ähnliche Wörter. Die Resultate scheinen aber zu hoch. Vor allem die Feature Matrix und die Covington Matrix erzeugen sehr viele Werte über 0.5, was meiner Ansicht nach viel zu hoch ist.

4.2.3 Erkennen von unvollständigen Wörtern

Bei diesem Versuch will ich beobachten, wie fest ein Wort abgeändert werden kann, bis es nicht mehr richtig erkannt wird. In der Realität entspricht dies einer undeutlichen Aussprache oder einem Dialekt.

Versuch:

Es soll das Wort ‚Kaffeemaschine' solange abgeändert werden, bis es nicht mehr aus der kompletten Wortliste erkannt wird. Dabei soll untersucht werden, welches Ähnlichkeitsmass das ursprüngliche Wort noch am besten erkennt.

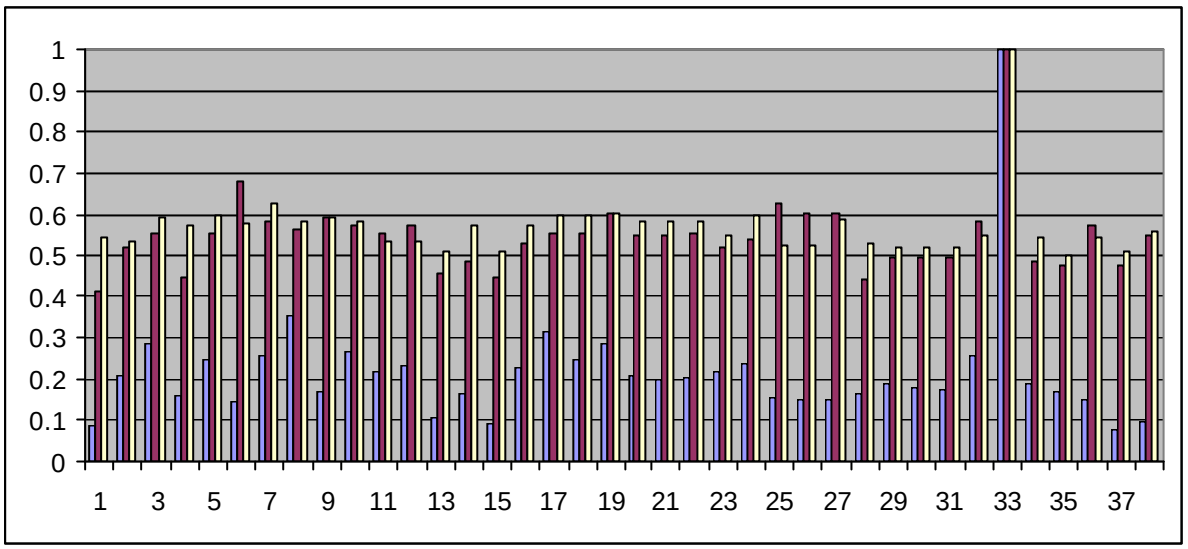

Abb. 22: WSK-Werte für die Erkennung von 'k a f e: m a S i: n @'

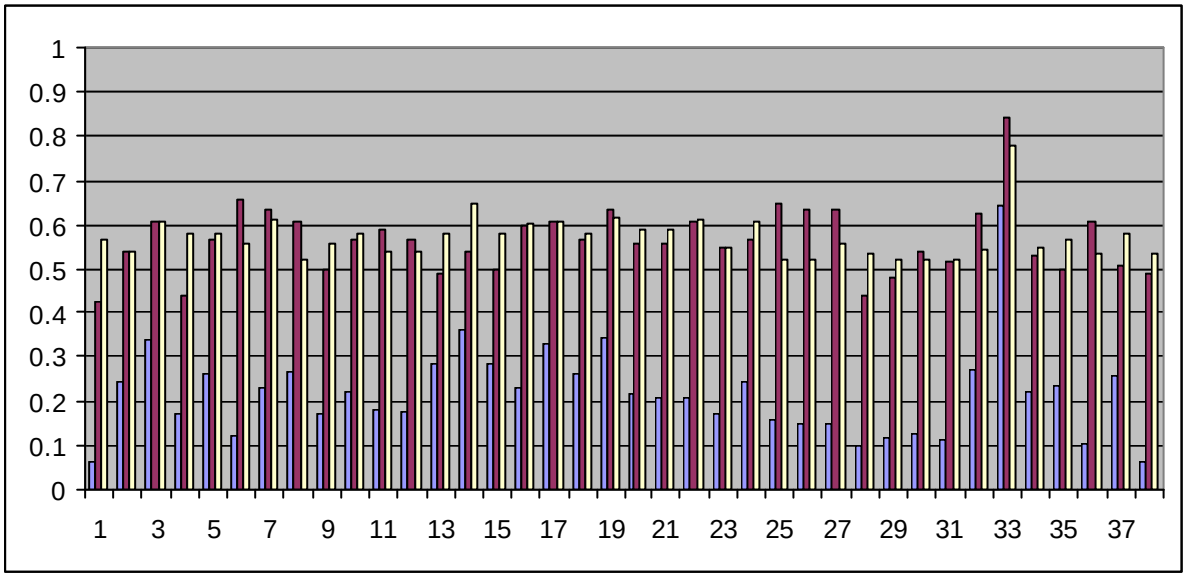

Abb. 23: WSK-Werte für die Erkennung von 'g a f E: m a: s I n'

Im Anhang B kann man die zu den Nummern gehörigen Wörter nachschlagen.

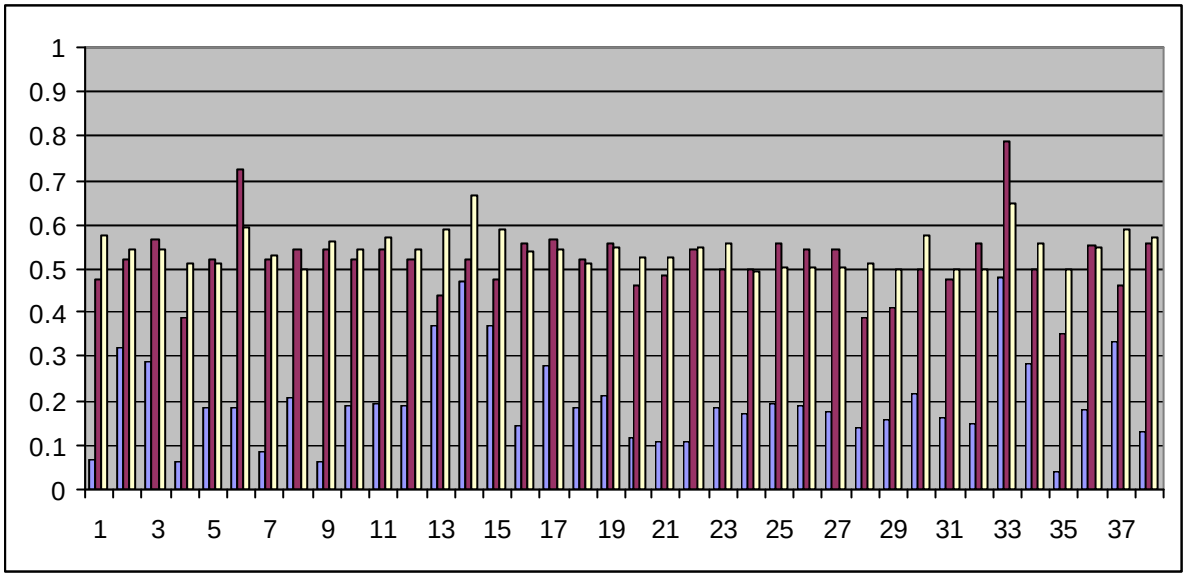

Abb. 24: WSK-Werte für die Erkennung von 'g a v E: m a: s @'

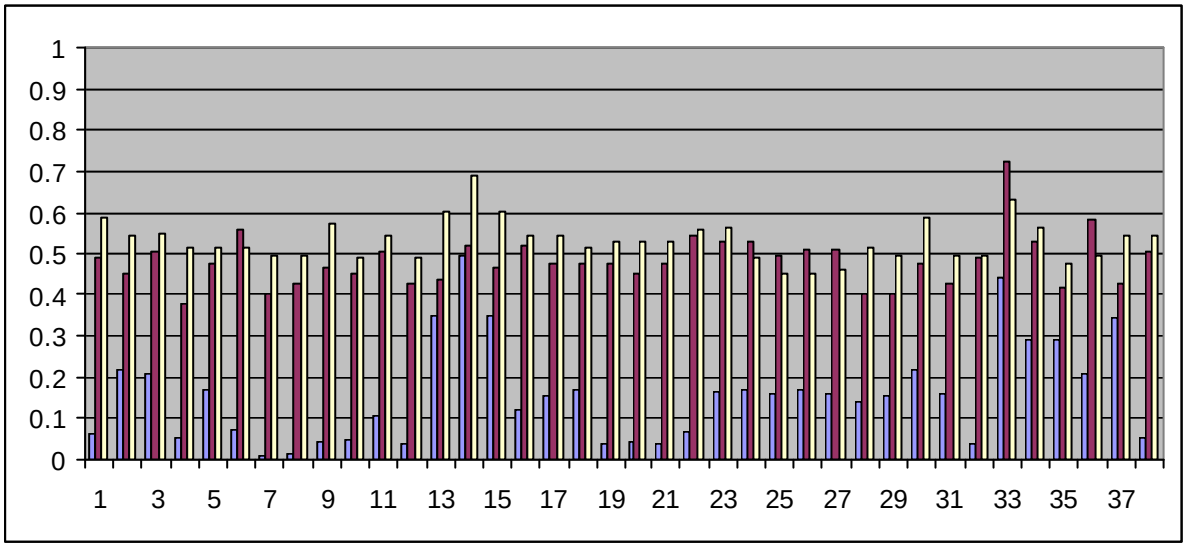

Abb. 25: WSK-Werte für die Erkennung von 'a v E: m a: s l'

Resultate:

Bei den ersten beiden Abänderungen (Abbildung 22 und 23) erkennen alle drei Ähnlichkeitsmasse noch das richtige Wort. Die Confusion Matrix liefert dabei die deutlichsten Unterschiede zwischen dem korrekten Wort ‚Kaffeemaschine' und den anderen Wörter der Wortliste. Nach der letzten Änderung erkennt nur noch die Feature Matrix das richtige Wort. Das Interessante daran ist, dass Covington und Confusion Matrix beide das gleiche falsche Wort wählen, nämlich ‚Maus' (Nr. 14).

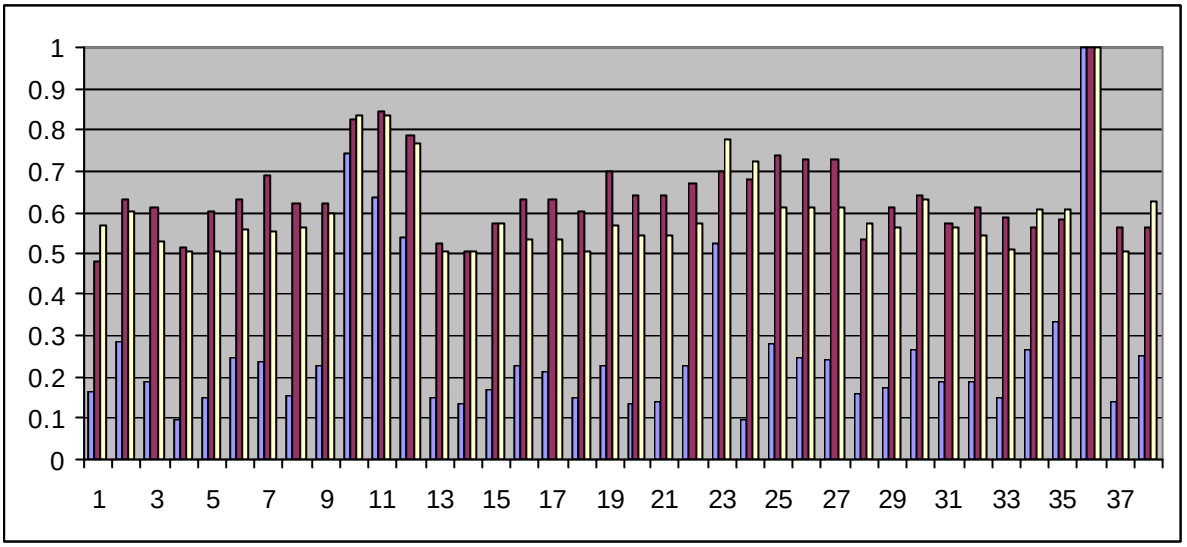

Abb. 26: WSK-Werte für die Erkennung von 'f E r v a l t @ r'

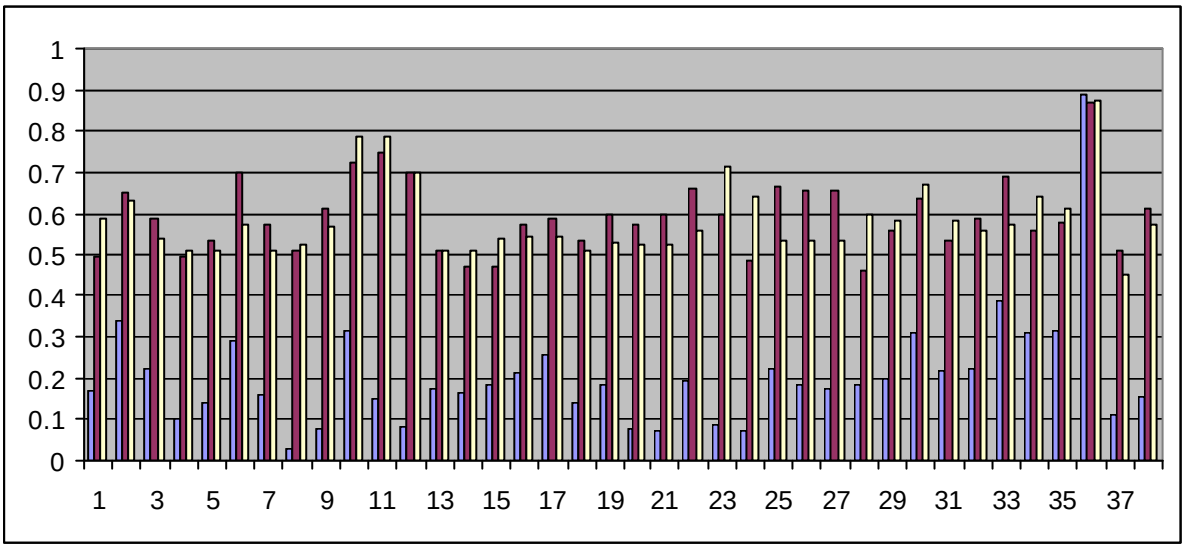

Abb. 27: WSK-Werte für die Erkennung von 'f E v a l t @'

Resultate:

Wie schon bei den vorherigen Resultaten liefert uns die Confusion Matrix die grössten Differenzen zwischen den Werten. Trotz einer undeutlichen Aussprache wird das Wort für alle drei Ähnlichkeitsmasse richtig erkannt.

4.3 Zusammenfassung der Beobachtungen

Abschliessend zu den Versuchen kann ich sagen, dass die beiden neuen Ähnlichkeitsmasse korrekt zu funktionieren scheinen. Die Confusion Matrix liefert dabei die besten Resultate. Ähnliche und unterschiedliche Wörter können mit der Confusion Matrix am besten unterschieden werden. Sie liefert auch ungefähr die Werte, die man intuitiv erwarten würde. Auch die Feature Matrix liefert zufriedenstellende Ergebnisse, wobei aber die Werte im Durchschnitt zu hoch ausfallen. Dies könnte mit einer detailierteren Implementierung und Normierung sicher noch verbessert werden. Das Covington Mass ist zu ungenau für den alleinigen Einsatz. Die Grundwerte sind sicher in einem vernünftigen Bereich, aber die fehlende „Auflösung“ schränkt die Aussagekraft dieses Masses sichtlich ein.

5 PROBLEME UND AUSSICHTEN

Die grössten Probleme hatte ich bei der Bestimmung der Normierungsgrenzen. Zum Teil erkannte ich falsche Normierungsgrenzen erst am Ende der Versuchsreihe und dann war die ganze Arbeit umsonst. Zuerst habe ich fixe Normierungsgrenzen gewählt, die nur von der Länge abhängig waren und nicht spezifisch auf die im Wort enthaltenen Laute eingingen. Dies führte zum Teil zu akzeptablen Lösungen. Schliesslich habe ich eine Normierung gewählt, die alle theoretisch möglichen Fälle berücksichtigt und vernünftige Werte zurückgibt. Der Maximalwert des Normierungsbereichs ist zum Beispiel so aufgebaut, dass er Wörter mit lauter Konsonanten oder Vokale berücksichtigt, obwohl diese in der Praxis nicht existieren. Dies führt dazu, dass die Werte für den maxVal meistens zu gross sind und dadurch zu hohe Wahrscheinlichkeitswerte resultieren. Man könnte also durch eine praktischere Lösung die Wahrscheinlichkeitswerte noch genauer berechnen.

Ein weiteres Problem war, dass ich nicht genau wusste was ich an den Ähnlichkeitsmassen untersuchen soll. Mangels Erfahrung in der Sprachverarbeitung wusste ich nicht welche Aspekte für eine Untersuchung interessant sein könnten und welche belanglos sind. Ich habe mich nun auf praktische Aspekte beschränkt und wollte in erster Linie zeigen, dass meine Implementierung des Algorithmus und der Ähnlichkeitsmasse funktionieren. In Zukunft könnte man natürlich diese Masse genauer unter die Lupe nehmen und so besser Vorteile und Mängel aufdecken.

Ein interessanter Ansatz für eine weiterführende Arbeit wäre das Verfeinern der Feature Matrix. Momentan sind die einzelnen Features noch nicht gewichtet. Es wird bloss die Anzahl unterschiedlicher Features gespeichert. Wenn man die Zusammenhänge der Features untersuchen würde, könnte man so vielleicht auf eine sinnvolle Gewichtung stossen, die dieses Ähnlichkeitsmass deutlich verbessern würde.

6 SCHLUSSWORT

Im Nachhinein blicke ich auf eine interessante Semesterarbeit zurück, welche mir einen Einblick in einen, für mich völlig unbekannten, Bereich der Technik ermöglichte. An dieser Stelle möchte ich mich auch bei meinem Betreuer René Beutler bedanken, der mir bei Problemen weitergeholfen hat und mich tatkräftig bei meiner Arbeit unterstützte.

A DIE LAUTLISTE

1	a	hat	[hat]	20	m	Mast	[mast]
2	a:	Bahn	[ba:n]	21	n	Naht	[na:t]
3	a_I	weit	[va_It]	22	N	lang	[laN]
4	a_U	Haut	[ha_Ut]	23	o:	Boot	[bo:t]
5	b	Ball	[bal]	24	O	Post	[pOst]
6	C	ich	[IC]	25	ox:	Öl	[ox:l]
7	d	dann	[dan]	26	oe	göttlich	[goetliC]
8	e:	Beet	[be:t]	27	O_Y	Heu	[hO_Y]
9	E	hätte	[hEt@]	28	p	Pakt	[pakt]
10	E:	wähle	[vE:l@]	29	r	Rast	[rast]
11	@	halte	[halt@]	30	s	Hast	[hast]
12	f	Fass	[fas]	31	S	schal	[Sa:l]
13	g	Gast	[gast]	32	t	Tal	[ta:l]
14	h	hat	[hat]	33	u:	Hut	[hu:t]
15	i:	viel	[fi:l]	34	U	Pult	[pUlt]
16	I	bist	[bIst]	35	v	was	[vas]
17	j	ja	[ja:]	36	x	Bach	[bax]
18	k	kalt	[kalt]	37	y:	Rübe	[ry:b@]
19	l	Last	[last]	38	Y	füllt	[fYlt]

B TESTWÖRTER-SET

1	'E C o:'	Echo
2	'h a t'	hat
3	'b a: n'	Bahn
4	'h o: f'	Hof
5	'd e: n'	den
6	'b @ h E n d @'	behände
7	'k E t s C @ n'	Kätzchen
8	'b a_U m S t a m'	Baumstamm
9	't O m b O l a'	Tombola
10	‘f a l t @ r’	Falter
11	’v a l t @ r’	Walter
12	’h a l t @ r’	Halter
13	‘h a_U s’	Haus
14	’m a_U s’	Maus
15	’l a_U s’	Laus
16	'd E n'	denn
17	'd a n'	dann
18	'l a: k @ n'	Laken
19	'l O k @ n'	Locken
20	'l Y k @ n'	Lücken
21	'l E k @ n'	lecken
22	'a l t @ r'	Alter
23	'a l t @ r n'	altern
24	't a_U s @ n d @ r'	Tausender
25	'l a_U s @ n d @ r'	lausender
26	'm a_U s @ n d @ r'	mausender
27	'U n t'	und
28	'h U n t'	Hund
29	'v U n t'	wund
30	'b U n t'	Bund
31	'b a_U m a r k t'	Baumarkt
32	'k a f e: m a S i: n @'	Kaffeemaschine
33	'a: l'	Aal
34	'f O_Y @ r t s O_Y k'	Feuerzeug
35	'f E r v a l t @ r'	Verwalter
36	'h o: x h a_U s'	Hochhaus
37	'r y: b @'	Rübe

QUELLENVERZEICHNIS

<Q> *Literatur*

Grzegorz Kondrak, *Alignment of Phonetic Sequences*

Technical Report CSRG-402, Department of Computer Science, University of Toronto

Dr. Beat Pfister, *Sprachverarbeitung I*

Vorlesungsskript, Institut für technische Informatik, ETH Zürich

<Q> *Hyperlinks*

Dynamische Programmierung

http://page.mi.fu-berlin.de/~benav/AlgoDS/dynprogr.pdf