2.6 Testbett

2.6.1 Erkennung mit HVite

Wir sind jetzt soweit, daß wir das erste Mal eine Erkennung starten können. Die initialisierten Modelle sind zwar bestimmt noch nicht ideal trainiert, aber formal können sie in einem Erkenner verwendet werden. Der nächste Schritt ist also der Aufbau eines sog. 'Testbetts', in dem der aktuell verfügbare Erkenner anhand des Test-Sets beurteilt werden kann.

Der eigentliche Erkennungs-Prozess wird vom Programm HVite gesteuert. Was HVite macht, ist kurzgefasst folgendes:

- Liest die Liste der zu verwendenden HMMs (Phonem-Modelle) und ein oder mehrere MMFs, und überprüft ob alle Modelle auf der Liste auch wirklich vorhanden und formal in Ordnung sind.
- Liest ein Aussprache-Lexikon ein und prüft, ob alle verwendeten phonetischen Modell-Namen in der Liste der verwendeten HMMs vorkommen.
- Liest ein SLF (Standard Lattice Format) Lattice ein und prüft ob alle Wörter in dieser Lattice auch im Lexikon definiert sind.
- Ersetzt die Wörter in der Lattice durch die entsprechenden Phonem-Folgen, wie sie im Aussprache-Lexikon definiert sind.
- Ersetzt die Phoneme in der Lattice durch die entsprechenden HMM und bildet so die entgültige Such-Lattice für den Token- Passing-Prozess.
- Liest ein Skript-File mit den zu erkennenden Signal-Files ein und prüft, ob diese kompatibel zur Modellstruktur sind.
- Verarbeitet ein Signalfile nach dem anderen, führt die Viterbi-Suche in der Lattice durch und schreibt das Ergebnis - also die erkannte Wortfolge - in ein Master Label File (MLF)

In Ihrem Gruppenverzeichnis befindet sich ein Signalfile aus dem Verbmobil Korpus und das zugehörige htk-File:

```
j514axx0_005_ULP.nis j514axx0_005_ULP.htk
```

Hören Sie sich das Signal an:

```
% test_nist p=1 j514axx0_005_ULP.nis
```

Der Aufruf für die Erkennung einer einzelnen Äußerung, die im File `j514axx0_005_ULP.htk` in Ihrem Gruppen-Directory gespeichert ist, lautet:

```
% HVite -H hmm1/VM.mmf -w LAT -s 6.5 -t 100 \
? -A -T 1 TESTDEV.lex PHONEME j514axx0_005_ULP.htk
```

HVite hat sehr viele Optionen und Argumente. Die allerwichtigsten sind:

Optionen:

```
-H nam : Filename eines MMF mit trainierten HMM-Modellen
-w nam : Filename des Lattice Files
-s #   : Gewicht des Language Models gegenüber der Akustik
         Dies ist ein linearer Faktor, mit dem die logarithmierte
         Wahrscheinlichkeit des skaliert wird; -s 1.0 hat keine Wirkung
-A     : Wiederhole den Befehl auf der Kommandozeile (Logging)
-T 1   : Trace level 1
```

Weitere Optionen (hier nicht verwendet):

```
-t #   : Breite des sog. 'beams' in logarithmierter Wahrscheinlichkeit
         Dieser Faktor bewirkt in Endeffekt, daß in jedem Zeitschritt
         nur Such-Tokens überleben, deren aufakkumulierte log.
         Wahrscheinlichkeit innerhalb eines Bereichs von # unter
         dem besten Token liegen.
-i nam : Filename des Output-MLF. Wird diese Option angegeben, werden
         keine rec Files erzeugt, sondern die Ergebnisse in ein
         gemeinsames MLF 'nam' geschrieben.
-o str : Restriktion des Outputs. Im allgemeinen braucht man nicht so
         viel Information. Deshalb kann man durch diese Option den Output
         einschränken. Typisch ist str=ST
-S nam : Filename des 'Skript' Files mit den zu erkennenden Input-Files
         (bei uns: ../DEV.slist)
-p #   : 'Word End Penalty' Der Wert # wird immer beim Verlassen
         eines Wortes auf die log. aufakkumulierte Wahrscheinlichkeit
         des Such-Tokens addiert. Negative Werte bedeuten also eine
         Bestrafung; positive eine Belohnung (Worteinfügungen werden
         gefördert); -p 0.0 hat keine Wirkung
```

Argumente:

```
TESTDEV.lex          : Aussprache-Lexikon
PHONEME              : Liste der verwendeten HMM
j514axx0_005_ULP.htk : Signalfile mit der unbekannten Äußerung
                       (bereits vorverarbeitet oder noch Signal)
```

Das Ergebnis der Erkennung wird in ein Label-File `j514axx0_005_ULP.rec` geschrieben.

```
% less j514axx0_005_ULP.rec
```

Was der Sprecher wirklich gesprochen hat, können wir im Wort-Referenzfile `TEST.REF.mlf` nachschauen

```
% less DEV.REF.mlf
/j514axx0_005_ULP
q
```

2.6.2 HResults: Messung der Wort-Akuratheit (word accuracy)

Wir wollen nun dieses Ergebnis dahingehend beurteilen, wieviele Wörter der Erkenner richtig erkennt ('Wort-Akuratheit', word accuracy) . Das prinzipielle Vorgehen dabei ist:

- bilde eine optimale Zuordnung von Wörtern aus dem Referenz-Satz und dem erkannten Satz, so daß die Summe aus ersetzten, eingefügten und gelöschten Wörtern minimal ist
- dann berechne die Wort-Akuratheit zu

$$A\,[\%] = \frac{N - S - D - I}{N}$$

mit:
- N : Anzahl der Wörter im Referenzsatz
- S : Anzahl Ersetzungen
- D : Anzahl Löschungen
- I : Anzahl Einfügungen

Das HTK-Tool HResults erledigt das für uns:

```
% cd ~/HTK-KURS/GRUPPE#
% HResults -t -I DEV.REF.mlf \
? TESTDEV.words j514axx0_005_ULP.rec
```

Optionen:

```
-t     : gib für jede Abweichung ein Transkript der Abbildung aus
-I nam : Filename des Wort-Referenz-MLFs
-e m n : Bilde das Wort 'm' auf 'n' ab, bevor die Abbildung gemacht
         wird. Diese Option dient dazu, 'Wörter' wie '!EXIT', die es
         ja in der Wortreferenz gar nicht geben kann, auf die sog.
         'Null-Klasse' '???' abzubilden. Auch andere Abbildungen
         sind möglich, z.B. 'is' auf 'ist', etc.
```

Argumente:

```
TESTDEV.words        : Liste aller vorkommenden Wörter
j514axx0_005_ULP.rec : Ergebnis-File (hier kann auch ein MLF stehen)
```

Schauen Sie sich den Output genauer an. HResults liefert uns bei diesem Aufruf nicht nur das prozentuale Ergebnis, sondern auch die Abbildung, die es verwendet hat. Beim genauen Hinschauen sehen wir, daß unser Ergebnis aus folgendem Grund nach unten verfälscht wurde:

Im REC-Teil sind zusätzlich die 'Wörter' '!ENTER' und '!EXIT' enthalten (resultiert in 2 Fehlern!). Der Grund hierfür ist, daß die Such-Lattice immer über diese beiden Pseudo-Wörter betreten bzw. verlassen werden muß. Akustisch sind beide 'Wörter' eigentlich Pausenmodelle (siehe Lexikon!).

Das Problem können wir umgehen, indem wir solche Wörter 'ohne Semantik' zum 'Null-Wörtern' erklären. Das bedeutet, daß HResults sie bei der Auswertung nicht beachtet.

```
% cd ~/HTK-KURS/GRUPPE#
% HResults -t -I DEV.REF.mlf \
? -e '???' \!ENTER \
? -e '???' \!EXIT \
? -e '???' \!SIL \
? -e '???' \!NOISE \
? -e '???' \!ANOISE \
? -e '???' \!BREATH \
? -e '???' \!LAUGH \
? TESTDEV.words j514axx0_005_ULP.rec
```

Und schon könnte die Erkennungsleistung besser geworden sein! (Manchmal auch nicht!)

Um zu testen, ob es sich bei der Differenz der Wort-Akuratheit nur um ein Zufallsereignis handelt, können wir die Signifikanz, d.h. den Wahrscheinlichkeitslevel für ein Zufallsergebnis berechnen lassen:

```
% signif -a <Acc1> -b <Acc2> -n <N>

     mit:   Acc1 : Wort-Akuratheit im ersten Versuch
            Acc2 : Wort-Akuratheit im zweiten Versuch
            N    : Anzahl der Einzeltests (Wörter)
```

Im Allgemeinen spricht man ab einer Signifikanz von 0.005 von einem sicheren Ergebnis.

Die obigen Beispiele verarbeiten nur ein einzelnes File. Natürlich ist das nicht sehr aussagekräftig. Wir können aber HVite und HResults auch in einer Form verwenden, in der sie die Erkennung des ganzen Development-Sets durchführen und beurteilen.

```
% cd ~/HTK-KURS/GRUPPE#
% HVite -H hmm1/VM.mmf -w LAT -s 6.5 -o ST \
? -t 100.0  -p -2.0 -S ../DEV.slist \
? -i $$OUTPUT.mlf -A -T 1 TESTDEV.lex PHONEME
% HResults -I DEV.REF.mlf \
? -e '???' \!ENTER \
? -e '???' \!EXIT \
? -e '???' \!NOISE \
? -e '???' \!SIL \
? -e '???' \!ANOISE \
? -e '???' \!BREATH \
? -e '???' \!LAUGH \
? TESTDEV.words $$OUTPUT.mlf
```

2.6.3 Kombiniertes Test-Shellskript

Da sich soviele Befehle nur noch schwer von der Tastatur eingeben lassen, empfiehlt sich auch hier die Verwendung eines Shell-Skripts.

Gehen Sie in einen Editor Ihrer Wahl und editieren Sie ein Shell-Skript 'TEST', welches die obigen Befehle zur Erkennung UND Beurteilung des Test-Sets ausführt. (Richten Sie sich an dem bereits vorhandenen Shell-Skript 'INIT'. Vergessen Sie nicht, dem fertigen Skript mit dem Befehl 'chmod

755 TEST' Ausführungsrechte zu geben.) Ersetzen Sie in Ihrer Skript-Version `hmm1/VM.mmf` durch `$1`. Dann können Sie mit diesem Skript in Zukunft verschiedene Modell- Sets prüfen indem Sie z.B. einfach eingeben:

```
% TEST hmm3/VM.mmf
```

Testen Sie das Skript bis zur nächsten Doppelstunde. Prüfen Sie vor dem Start, ob auf Ihrem Rechner schon andere Prozesse laufen:

```
% top
```

Wenn die 'load averages' (erste Zeile) 1.0 oder hoher sind, sollten Sie keinen weiteren Prozess starten. Starten Sie ihr Skript im Hintergrund und lenken Sie stdout und stderr in ein Log-File um:

```
% nice ./TEST hmm1/VM.mmf >& TEST_hmm1.log &
```

Dann können Sie sich gefahrlos ausloggen und der Prozess läuft im Hintergund weiter. Sie können jederzeit das Logfile abfragen mit

```
% tail -f TEST_hmm1.log oder less TEST_hmm1.log
```

und mit Ctrl-C wieder abbrechen, ohne Ihren Prozess TEST zu beeinflussen.

Damit haben wir jetzt eine erste Basis, was unser Erkenner minimal zu leisten vermag. Alle weiteren Schritte (was auch die Mächtigkeit von HTK ausmacht!) ist die Verbesserung dieser Leistung durch die verschiedensten Verfahren.

2.6.4 Freiwillige Zusatzübungen

Ihr Evaluationsskript hat ein File `<nummer>OUTPUT.mlf` mit den Ergebnissen des Erkenners erzeugt. Lassen Sie HResults noch einmal darüber laufen, aber ohne die Wörter 'ohne Semantik' auf '???' abzubilden. Was für eine Word Accuracy ergibt sich jetzt?

Machen Sie eine Kopie des Lexikons `TESTDEV_ohne_single.lex` und löschen Sie darin alle 'Wörter', die nur aus einem Laut bestehen, z.B. 'k k' Dann lassen Sie Ihr Testbett mit diesem neuen Lexikon laufen und vergleichen die Ergebnisse. Löschen Sie nun auch alle zwei-lautigen 'Wörter', die keine Bedeutung haben, z.B. 'di'. Lassen Sie aber 'echte' Wörter' im Lexikon, z.B. 'an' und 'am'. Wiederholen Sie dann den Test und diskutieren Sie die Ergebnisse. Sind die Unterschiede überhaupt signifikant?