.A gasztroszkópiai és ultrahangleletező beszédfelismerő leírása

Műszaki dokumentáció

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Távközlési és Médiainformatikai Tanszék,
Beszédakusztikai Kutatólaboratórium

A kutatás és fejlesztés az OTKA T 046487 ELE, és az IKTA 00056 pályázatok keretében történt.
2006

1. fejezet – szerkezeti felépítés

A futtatható beszédfelismerő végleges formájában három külön programból áll össze. Ezek a következők:

1. *Akusztikai Markov-modell generáló program*
2. *Nyelvi Bigram-mező generáló program*
3. *Valós idejű beszédfelismerő szoftver*

Ezek a programok kihatással vannak egymás működésére: az akusztikai szint befolyásolja a felismerés minőségét, a nyelvi szint a felismerő nyelvtani bonyolultságát, míg a felismerő eredménye visszahat(hat) a nyelvi szintre. Ez lehetővé teszi a nyelvtan folyamatos bővülését, adaptálódását a megfelelő tartalomhoz.
A következőkben egyenként áttekintjük a megfelelő programok belső szerkezetét, a bennük használt matematikai modelleket és a szoftveres megvalósításaikat. Mivel az akusztikai modellek létrehozása (és a hozzá tartozó szoftver elkészítése) a projekt 2004-es évében megtörtént, ezért erre külön nem térünk ki.

2. fejezet – nyelvi Bigram-mező generáló program

Ebben a fejezetben áttekintjük a nyelvi motor tulajdonságait a matematikai háttértől kezdve az adatok tárolásának módjáig.

2.1. Nyelvi modell

A felismerő a nemzetközi vonalon elfogadott és használt n-gram modellt alkalmazza a nyelvi szint definiálásához. Ismeretesek ennek előnyei és hátrányai, a magyar nyelvre való illesztés problémái és a kötetlenebb nyelvtan nehézkes modellezése. Ennek ellenére a feladatban megfogalmazott jelentősen kötött szótári és nyelvtani elemek miatt mégis bigram modell megvalósítását tűztük ki célul.
Az n-gram modellekben a nyelvi modellek szószekvenciáik valószínűségének halmazából áll.
A szekvencia valószínűsége ekkor:

$$P(w_1, w_2, ...w_m) = P(w_1)\prod_{i=2}^{m} P(w_i | w_{i-1} ... w_1)$$

A kontextust limitálva:

$$P(w_1, w_2, ...w_m) \cong P(w_1)\prod_{i=2}^{m} P(w_i | w_{i-1} ... w_{i-n+1})$$

ahol n>0 tetszőlegesen választott konstans egész. A nyelv olyan tulajdonságokkal rendelkezik, hogy a folyamat során egy későbbi állapot valószínűsége gyakorlatilag független a kezdőfeltételektől, így n értékére nem kell nagy n értéket használni. (Tipikus értékek 2-től 6-ig)
A fenti valószínűség ekkor a következőképpen számítható ki:

$$P(w_i | w_{i-1} ... w_{i-n+1}) = \frac{N(w_i ... w_{i-n+1})}{N(w_{i-1} ... w_{i-n+1})}$$

ahol N(.) a megadott szekvencia előfordulásai száma a tanító szöveganyagban.

2.2. Programszerkezet

A nyelvi szint létrehozásához egy szekvenciális menetű segédprogram készült, melynek csupán bemenetei és kimenetei van definiálva, ezek a következők:
Bemenetek:

- *eredeti Bigram-modell*
- *szótárkészlet*
- *mintaanyag*

Kimenetek:

- *módosított Bigram-modell*
- *módosított szótárkészlet*
- *szólista*

– *az összes eddig használt mintaanyagot tartalmazó szövegfájl*

A program működése a következő:

1. *Mintaanyag beolvasása*
2. *Szótár beolvasása*
3. *Mintaanyag és szótár illesztése, a szótár bővítése*
4. *eredeti Bigram-modell beolvasása*
5. *Bigram-modell valószínűségek módosítása*
6. *Szótár mentése*
7. *Bigram-modell mentése*
8. *Mintaanyagok mentése*
9. *Szólista mentése*

A két utolsó művelet valójában szükségtelen a felismerő működése szempontjából, de hasznos információkat tartalmaz a felhasználó adminisztrátor részére.

2.3. Szótárillesztés

A szótárkészletet az aktuális mintaanyaggal össze kell vetni. Háromféle lehetőség van:

– *a szó szerepel a szótárban – nem kell tenni semmit*

– *a szó nem szerepel a szótárban – fel kell venni a szót a szótárba*

– *a szó nem szerepel a szótárban, de létezik olyan szó, mely ennek szinonímája, rövidítése – csere definiálása*

Az első esetben értelemszerűen nem szükséges semmilyen beavatkozás.
A második esetben a megfelelő szót fel kell venni a szótárba. Ehhez kell a szó kiejtett alakja („K”), amit a felhasználónak kell megadnia. Ezután a program UnitSAMPA Make(string); függvényének hívásával létrehozza a sampa átiratot. Ekkor már mindhárom szükséges adat rendelkezésre áll, így a szó felvehető a szótárba.
A harmadik esetben a felhasználónak rendelkeznie kell, hogy mely szóra cseréli az új ismeretlen szót. Ehhez a program által adott ajánlatokból tud választani egyet. Figyelem! Ha nincs a választhatók között nekünk megfelelő, akkor a szót fel kell venni! A program a választható szavakat a következőképpen választja ki:

– *az összes egy karakteres szó (rövidítés)*

– *az összes olyan szó, melnyek Levehnstein távolsága kisebb, mint 2*

– *az olyan szavak, melyek csak írásjelben különböznek*

A Levehnstein távolságot az LD és LD2 függvények segítségével határozhatjuk meg.
Miután az új szavak illeszkedtek, az új mintaanyagban kicserélődik az összes csereszó. Ezt követően pedig kezdetét veszi a bigram-mező feltöltése.

2.4. Bigram-mező létrehozása

3. Adatszerkezet

A továbbiakban áttekintjük a program által használt adatok szerkezeti formáit, a fájlok struktúráit.

3.1. Szótár

A szótár tartalma a program könyvtárában elhelyezett words.csv fájl. Ez a fájl egy CSV formátumú adatsort tartalmaz szöveges formában. A szeparáló jel az ASCII-ban használatos függőleges | karakter. Az adathalmaz fejléce a következő: L|K|S, melynek jelentése: leírt alak|kiejtett alak|sampa alak. A leírt és kiejtett alak egyértelmű, a magyar karakterekkel helyesen- és kiejtett formában leírt szavakat tartalmazza. A sampa alak a szavak sampa karakterekkel való leírását tartalmazza, figyelembe véve, hogy a felismerő milyen sampa karaktereket alkalmaz, és azoknak mi a jelölése.
Figyelem! A felismerő által használt karakterek jelölése eltérhet a szabványos sampa jelölésrendszertől! Az illeszkedés ellenőrzése mindenkor az adminisztrátor feladata.

3.2. Bigram-mező fájl

A bigram-mező a felismerő által használt legbonyolultabb fájlstruktúra, ezért teljes részletességgel kell bemutatni.
A fájl egy tömörített adatsor, melyet a Borland Delphi zLib könyvtárának streamje segítségével tömörítünk.
A fájl tartalma:

Elnevezés	***Típus***	***Értelmezés***
Num	*4 byte integer*	*Szavak száma*
Num2	*4 byte integer*	*Markov-állapotok száma*
Size1	*8 byte integer (int64)*	*CS hossza*
CS	*StringList*	*Teljes tanítóanyag*
Size2	*8 byte integer (int64)*	*CS1 hossza*
CS1	*StringList*	*Szótár cserélendő szó leírt alak*
Size3	*8 byte integer (int64)*	*CS2 hossza*
CS2	*StringList*	*Szótár csereszó leírt alak*
Size4	*8 byte Integer (int64)*	*SL hossza*
SL	*StringList*	*Szavak*
Size5	*8 byte Integer (int64)*	*SL2 hossza*
SL2	*StringList*	*Sampa karakterek száma*
Size6	*8 byte Integer (int64)*	*SL3 hossza*
SL3	*StringList*	*Sampa karakter helye a Markov-bigram mezőben*
Size7	*8 byte Integer (int64)*	*SL4 hossza*
SL4	*StringList*	*Sampa alak*
Size8	*8 byte Integer (int64)*	*SL4b hossza*
SL4b	*StringList*	*Kiejtett alak*
Size9	*8 byte Integer (int64)*	*SL5 hossza*
SL5	*StringList*	*Markov-állapotok sampa nevei*
X1	*4 byte Integer*	*Bigram X1. szó*
Y1	*4 byte Integer*	*X1. szó darabszám*
Si1	*Single*	*X1. szó előfordulási valószínűség*
Y2	*4 byte Integer*	*X2. Bigram utód-átmeneteinek száma*
X2	*4 byte Integer*	*Bigram X2. szó*
YY	*4 byte Integer*	*Utód YY. Szó (sorszáma)*
Z	*4 byte Integer*	*Átmenet darabszáma*
Si2	*Single*	*Átmenet előfordulási valószínűsége*

A sárgával jelölt adatok Num-szor szerepelnek az adatsorban.
A zölddel jelölt adatok szintén Num-szor szerepelnek az adatsorban, azon belül a pirossal jelölt adatsorok Y2-szer szerepelnek minden zölddel jelölt adatsor után.
Ahogyan az az adatszerkezetből is látszik, a Bigram-mező mátrixa egy egydimenziós listára épül. A lista minden eleméhez két további lánc csatlakozik: egyik megmondja, hogy mely állapotok következhetnek, a másik pedig, hogy milyen valószínűséggel. Ezzel az elrendezéssel csak az átmenetek számának kétszeresét kell eltárolni információként, míg a teljes mátrix az átmenetek számának négyzete. Ebből következik, hogy mindaddig hasznos memóriaszempontból ez a struktúra, amíg a modell átmeneteinek száma el nem éri a mátrix 50%-át. Jelenleg az átmenetek még az 1%-ot sem közelítik meg. (Sparse-mátrix) Az 50%-os kitöltöttség természetesen lehetetlenné teszi a felismerést már pár ezer szó esetén is, mivel a választható útvonalak száma óriási lesz.

3.3. Mintaanyag és egyéb szövegfájlok

Ezekben a fájlokban egy közös konvenció létezik: minden szó külön sorba íródik. Ezen felül a mintaanyag esetében speciális karaktereket különböztetünk meg:

- *_ „aláhúzás jel”: tiltja az előző és a következő szó között a bigram-valószínűség számlálását. Hasznos mondathatár, vagy bekezdések határának esetében, amikor nincs statisztikai jelentősége a folytatásnak, valamint a # jelek használatakor. (lásd alább)*
- *#+ és #- jelek: átjárhatóságot biztosítanak két független „A” és „B” mező között. A #+ jel után a „B” mező bármely tagja következhet és a „B” mező bármely tagja után következhet a #- jel. Ezzel visszatérhetünk az „A” mező megfelelő eleméhez. A #+ és #- jelek a szavakkal azonos módon működnek, de a leírt szövegben nem látszanak, akusztikai jelentésük az esetleges szünettartáson kívül nincsen.*
- *#1-#9 jelek: számok esetében az adott szó(szám) rangját jelöli, az akusztikában nem vesz részt, csak sorrendi okokból használatos. Egy példa: 1205=ezer#5 kettő#4 száz#3 öt#1*

4. fejezet – Beszédfelismerő szoftver

1. Markov-modell

Vegyünk alapul egy N lehetséges állapotból álló rendszert. A rendszer t=1,2..n diszkrét időpontokban egyik állapotból a másikba kerülhet, ahol minimum 1 időegységet (frame) tölt el. Reprezentálni a rendszert egy gráffal a következőképpen lehet: a gráf csomópontjai az állapotok, élei az állapotok közti átmenetek. Minden átmenetet engedélyezve olyan teljes gráfot kapunk, amelyben minden csomóponthoz létezik egy olyan él, amely önmagába záródik. Mátrixokra leképezve ez azt jelenti, hogy az $N \times N$ *-es mátrix minden pontjában értelmezve van egy szám. A szám nagysága a csomópontokat összekötő élek súlya, mely az állapotátmenet valószínűségét jelenti. A továbbiakban bevezetve a következő jelöléseket:*

- q_t*: az az állapot, amelyikben a modell a t időpillanatban van,*
- a_{ij}*: az i. állapotból a j. állapotba mutató él súlya, ahol* $\sum_j aij = 1$*,*
- $\pi_i = P(q_1 = i)$ *az i. állapot kezdési valószínűsége,*
- *valamint* $O = (o_1 o_2 ... o_r)$ *a vizsgált folyamat.*

Rendeljünk a csomópontokhoz valószínűségi értéket, úgy hogy az kimutassa annak valószínűségét, hogy a folyamat o_k *szimbólumát éppen az adott csomópont generálta. Ehhez első megközelítésben ismerni kell a szimbólumok S halmazát. Ebben az esetben egy B:* $|S| \times N$ *-es mátrixxal reprezentálni lehet a fenti valószínűségeket. Ekkor Markov-modellnek nevezzük a* $\lambda = (A, B, \pi)$ *paraméterekkel megadott objektumot. A modell működése igen egyszerű, ezért nem térek ki rá. Röviden be kell mutatni viszont, hogy milyen módon adhatók meg a valószínűségi értékek, és hogyan kell kiszámolni azokat. A modell egy tetszőleges* $q = (q\ q_2 ... q_T)$ *állapotsorozat mentén bocsátja ki az O sorozathoz megfelelő eseményeket. Annak a valószínűsége, hogy éppen azt az eseményt generálja a modell, mint amit O alapján elvárunk:*

$$P(O \mid q, \lambda) = \prod_{t=1}^{T} P(o_t \mid q_t, \lambda)$$

Ez átalakítások után a következő alakra hozható:

$$P(O \mid \lambda) = \sum_q P(O \mid q, \lambda) P(q \mid \lambda)$$

Kérdés, hogy hogyan számíthatók ki a jobb oldalon álló valószínűségek? A dinamikus programozásból ismert Fowrard-Backward algoritmus segítségével:

Az előre mutató valószínűségek számítása:

1. inicializáló lépés: ..... $\alpha_1(i) = \pi_i b_i(o_1)$

2. indukciós lépés:..... $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1})$

A hátra mutató valószínűségek számítása:

1. inicializáló lépés: ..... $\beta_T(i) = 1$

2. indukciós lépés: ..... $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$

Ezekkel az algoritmusokkal egy meglévő HMM vizsgálata végezhető el. Nem esett szó azonban a modellek előállításához szükséges algoritmus(ok)ról. Ehhez rendelkezni kell olyan eseményvektorokkal, melyekhez elő szeretnénk állítani egy Markov-modellt. A továbbiakban feltételezzük K számú O eseményvektor létezését, melyek mindegyikére előállítottuk a Forward-Backward algoritmus segítségével a valószínűségi együtthatókat. Ekkor egy lokális maximumhoz vezető utat biztosít a Baum-Welch (BWA) algoritmus (Baum, 1972). Két segédváltozóra szükség lesz a számítások folyamán:
Az első jelöli annak valószínűségét, hogy O generálása során a modell a t. pillanatban az i, t+1. pillanatban pedig a j állapotban volt:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

$$\xi_t^k(i,j) = \frac{P(q_t = i, q_{t+1} = j, O^k \mid \lambda)}{P(O^k \mid \lambda)} = \frac{\alpha_t^k(i) a_{ij} b_j(o_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t^k(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}^k(j)}$$

A második megmutatja, hogy O generálása során mekkora a valószínűsége, hogy a t. pillanatban az i. állapotban van a modell:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Ekkor az alábbi összefüggések írhatók fel:

$\sum_{t=1}^{T-1} \gamma_t(i) =$ *az i. állapotból induló átmenetek száma*

$\sum_{t=1}^{T-1} \xi_t(i,j) =$ *az i. állapotból j. állapotba menő átmenetek száma*

Baum-Welch algoritmussal a következő iteráció paramétereit kiszámolni a következő formula szerint lehet:

kezdőállapot valószínűségek: ..... $\bar{\pi}_j = \gamma_1(i)$

átmeneti valószínűségek: ..... $\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \xi_t^k(i,j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \gamma_t^k(i)}$

kibocsátási valószínűségek: ..... $\bar{b}_j(k) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{s,t,o_t=v_k}^{T} \gamma_t^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T} \gamma_t^k(j)}$

Ezekkel megoldottuk a véges S szimbólumkészlettel rendelkező modellek leírását. Ha az S szimbólumkészlet nem határozható meg egyértelműen, vagy S mérete túlontúl nagy (végtelen), akkor B értékeinek kiszámításához függvényeket kell értelmeznünk. Alapvetően két elfogadott módszer használatos:

- *vektorkvantálás (VQHMM)*
- *folytonos valószínűségi tér használata (CHMM)*

Az utóbbi módszer képletei hasonló felépítésűek, mint a fent megismertek. A valószínűség-sűrűség függvény ez esetben elliptikus függvények összegeként tárgyalható. Tipikus megvalósítás Gauss-eloszlások alkalmazása, melynek során a gaussi paramétereket kell beállítani a megfelelő értékre. A kibocsátási valószínűség ekkor a következő módon írható fel:

$$\bar{b}_j(o) = \sum_{k=1}^{K} c_{jk} N\left(o, \mu_{jk}, U_{jk}\right)$$

ahol N gaussi PDF (Probability Desinty Function). A gauss függvény általános esetben ekkor a következő alakot ölti:

$$\bar{b}_j(o) = \frac{1}{\sqrt{(2\pi)^D \det(U_j)}} \exp\left\{-\frac{1}{2}(o-\mu_j)^T U_j (O-\mu_j)\right\}$$

Látható, hogy a szórásokat mátrixként lehet reprezentálni. Általában csak a főátlóban lévő tagokkal szokás számolni, mivel a szűrősorok paramétereinek egymás közti kovarianciája igen kicsi. (Rabiner, Juang, 1993) Ekkor a következő végleges forma jelenik meg:

$$\bar{b}_j(o) = \sum_{k=1}^{M} c_{jk} \frac{1}{\sqrt{2\pi\sigma_{jk}}} \exp\left\{-\frac{(o-\mu_{jk})^2}{2\sigma_{jk}^{\ 2}}\right\}$$

A BWA a gaussi paraméterek kiszámításához is nyújt megoldást:

gauss-koefficiensek: ..... $\bar{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)}$

várható értékek: ..... $\bar{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) o_t}{\sum_{t=1}^{T} \gamma_t(j,k)}$

és szórásnégyzetek: ..... $\bar{U}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)(o_t-\mu_{jk})(o_t-\mu_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)}$

A képletekben feltüntetett segédfüggvény pedig:

$$\bar{\gamma}_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}N(o_t,\mu_{jk},U_{jk})}{\sum_{m=1}^{M} c_{jm}N(o_t,\mu_{jm},U_{jm})}\right]$$

Ezzel a Markov-modellek elméleti alapszintű bemutatásának végéhez értünk.

1.8. 2. Viterbi algoritmus

A legjobb útvonal keresése az a feladat, aminek megoldása során meghatározható, hogy adott eseményvektort a modell milyen állapotsorozata mentén bocsát ki a legnagyobb valószínűséggel. Ezt 'lágy' Viterbi-algoritmusnak (Viterbi, 1967) nevezzük, és annyiban különbözik a Forward algoritmustól, hogy a legvalószínűbb útvonalakat meg kell jegyezni:

1. segédváltozók inicializálása: ..... $\delta_i(i) = \pi_i b_i(o_1)$, $\psi_1(i) = 0$

2. rekurziós lépés: ..... $\delta_t(j) = \max_{1\le i\le N}\left[\delta_{t-1}(i)a_{ij}\right] b_j(o_t)$, $\psi_t(j) = \arg\max_{1\le i\le N}\left[\delta_{t-1}(i)a_{ij}\right]$

3. terminációs lépés: ..... $P^* = \max_{1\le i\le N}\left[\delta_T(i)\right]$, $q_T^* = \arg\max_{1\le i\le N}\left[\delta_T(i)\right]$

4. útvonal visszafejtés: ..... $q_t^* = \psi_{t+1}(q_{t+1}^*)$

1.9. 3. Viterbi adatok tárolása, törlése

A folyamatos felismeréshez szükség van a Viterbi algoritmus végrehajtása során kiszámolt útvonalak időszakos vágására, különben a rendelkezésre álló memória nem lesz elegendő egy percnyi beszéd felismerésére sem. A Beam Search vágási értékét úgy kell megválasztani, hogy a valós időben történő felismerést lehetővé tegye ugyanakkor a lehető legtöbb variációt megvizsgálhassa az algoritmus. Ekkor - a mai számítógépek kapacitása mellett - átlagosan 5000-10000 állapot versenyez egymással. Minden útvonalpont tárolásához 8 bájt szükséges: 4 bájt a valószínűség leírásához, 4 pedig a helyzet jelöléséhez. Ilymódon másodpercenként

*7500*8*100 bájt=5.7 MB memória töltődik fel információval. Ilyen feltételek mellett az 512 MB memóriakapacitás (melyből az operációs rendszer a felét foglalja) kb. 45 másodperc alatt felemésztődik. Ezért van szükség mindenképpen a múltbeli útvonalak elhagyására.*

Az adatok eldobása a következő elv szerint zajlik:

Minden egyes Viterbi-algoritmus kiértékeléskor (200ms-onként) egy 10 másodperces szakaszt vizsgálunk, melynek első 20ms hosszú része lesz az eldobandó információ. (10 másodperc nem feltétlenül szükséges, de ebben az esetben már elenyésző a valószínűsége annak, hogy romoljon a felismerés valószínűsége az eldobás miatt.) A szakasz útvonalainak kiértékelése után az útvonalak gyökereit töröljük, de a valószínűség értékeket továbbra is azokból származtatjuk. Így azok hatása a versengő útvonalakra továbbra is megmarad.

*A fent leírt módszerrel akkor kerülhető el a memória telítődése, ha folyamatosan a régi helyekre töltődik fel az új információ. Ebben az esetben az aktuális hosszabb-rövidebb útvonalsorok mindig ugyanarra a helyre kerülnek. Ehhez azonban a 10 másodperces szakasz tökéletes lefedése kell, amelynek lényegesen több információ tárolásának lehetőségét kell magában tartania, mint amennyire valóban szükség van. (A Beam Search vágási kritériuma engedélyezi az adatok 'Burst'-ös elhelyezkedését, azaz lehetséges, hogy egy kiválasztott intervallumban az összes, vagy ahhoz közeli számú állapot verseng.) Így 10*100*N*8*3*8 bájtra, azaz ~1800 MB memóriára van szükség. (Ez esetben feltettük, hogy N=10000 szó és szavanként átlagosan 8 fonéma verseng az algoritmusban.) Ez a jelenlegi erőforrások mellett sajnos nem lehetséges, így azt a változatot kell választani, amikor a memóriában az adatokat az operációs rendszer által választott véletlen pozíciókban tároljuk. Ekkor megvan az esélye, hogy olyan memóriakiosztás áll elő, amely mellett az új adatsort csak a virtuális memóriát igénybe véve lehet elhelyezni, ez pedig a valós felismerés hamaros leállását vonja maga után. Az ilyen leállások valószínűsége igen kicsi, mivel azt egyre nagyobb méretű adatsorok tárolása okozhatja, az adatsorok mérete pedig felülről korlátos és azon belül valamilyen normálishoz közeli eloszlást követ. (Így egy új nagyméretű adatsor valószínűleg elfér egy régebbi, már törölt nagyméretű adatsor helyén.) A hiba előfordulása minimalizálható a felismerő hosszú szünetekben történő ismételt inicializálásával, amikor a memóriát felszabadítjuk és a felismerő processzt újraindítjuk.*

1.10. 4. Információk szinkronizálása

Ahhoz, hogy az eredményeket folyamatosan meg lehessen jeleníteni annak ellenére, hogy 200ms időközönként egy múltbeli 10 másodpercnyi intervallumot értékel ki a felismerő, szükség van a már kijelzett és a kijelzendő információk egyeztetésére. A karaktersorozatok esetében a következő alapvető lehetőségek merülnek fel:

Régi sorozat	*Új sorozat*	*Ok*	*Teendő*
abcd	*abcd*	*nincs új adat a 100ms alatt és nem is vesztettünk adatot t-10s időpontban*	*nincs*
abcd	*abcde*	*új adat jelent meg*	*y küldése*
abcd	*bcd*	*adat veszett el a t-10s időpontban*	*abcd törlése és bcd újraküldése*
abcd	*bcde*	*az előző két eset együttesen*	*abcd törlése és bcde küldése*

Az ilyen esetekben a felismerő nem változtatott a régi, már megjelenített információkon. Ezeken felül a következő speciális lehetőségek merülnek fel:

Régi sorozat	*Új sorozat*	*Ok*	*Teendő*
abcd	*abd*	*Változás történt az új 100ms közben*	*cd törlése és d küldése*
abcd	*bd*	*az kiértékelés változott, valamint a t-10s időpontban adatvesztés is történt*	*cd törlése d küldése*
abcd	*bde*	*új információ is megjelent*	*cd törlése és de küldése*

A fent vázolt eseteket a program lekezeli. Elméletileg elképzelhető, hogy a speciális esetek közötti második-harmadik esetben az 'a' információ nem a t-10s időpontban történő természetes adatvesztés miatt tünt el, hanem a felismerő változtatta meg a döntését. Annak az esélye azonban elenyésző (a teljes tanító- és tesztanyagban nem volt rá példa), hogy 10 másodperccel későbbi információ megváltoztassa a döntést, így ezt az esetet lekezelés nélkül lehet hagyni.

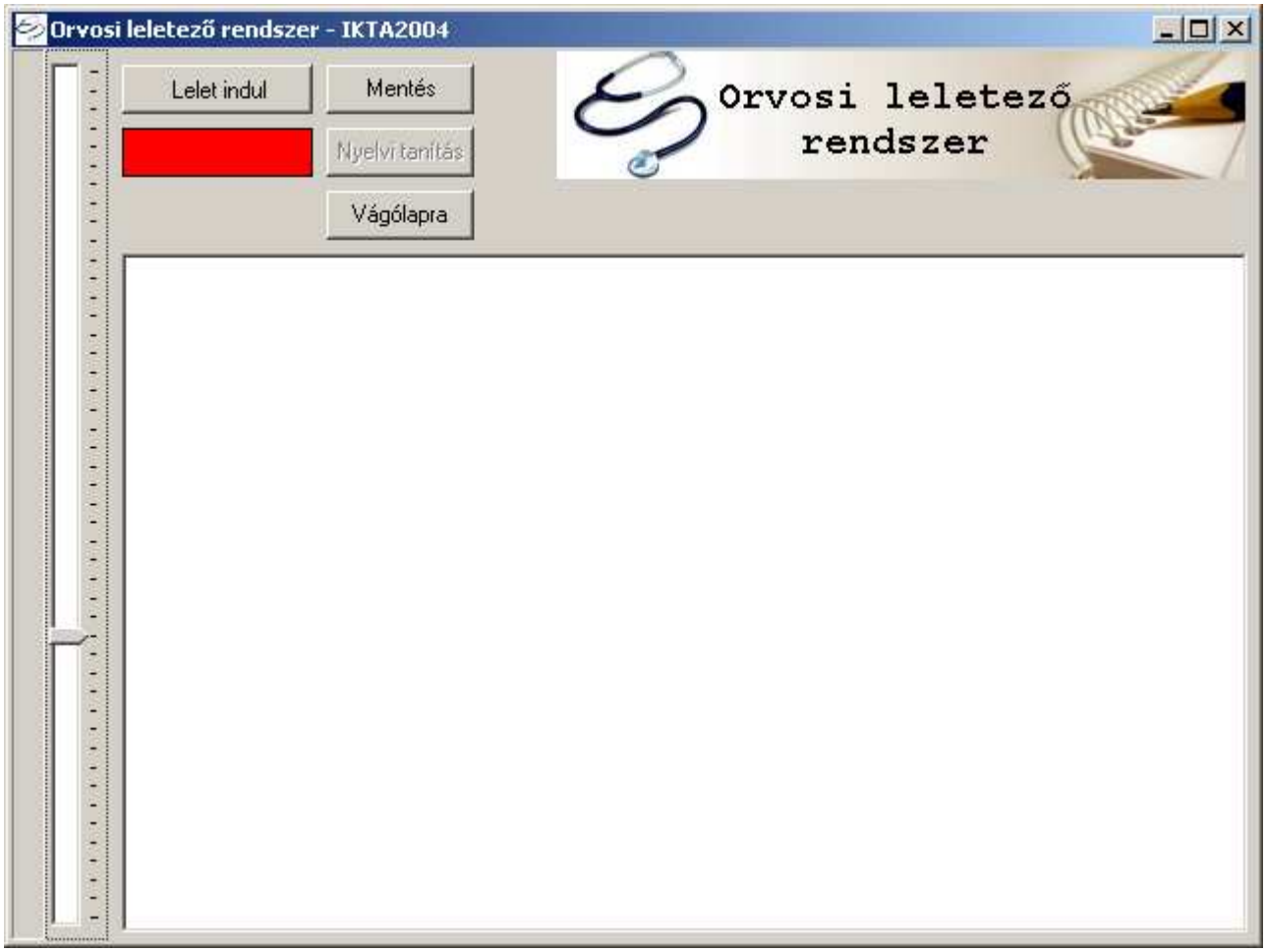
1.11. 5. Vizuális interfész

Az orvosoknak szánt végleges verzióban a felhasználóknak egyéni azonosítójuk van, amely segítségével a nekik megfelelő anyagot lehet kiválasztani.
Ezek után a mikrofon vételének ellenőrzése történik meg, probléma esetén a szoftver segítséget nyújt.

Ha teljesítettük a mikrofonnal kapcsolatos beállításokat, megjelenik a program főablaka, melyen a diktálást el lehet végezni. Az ablak bal oldalán található egy két oszlopból álló hangerőszabályozó. A bal oldali oszlop kék színű kitöltöttséggel jelzi az aktuális hangerőt, míg a jobb oldalival beállíthatjuk a kívánt küszöbértéket. Ennek hatására csak azon hangok fognak a felismerő bemenetére jutni, melyeknek hangereje meghaladja a küszöbértéket. Így zajos környezetben feljebb, csendes környezet esetén pedig lejjebb érdemes a csúszkát mozgatni. (Ha szükséges)
A felső gombsor környezetében található egy színes (zöld, vagy piros) téglalap. A téglalap zöld színe jelzi, hogy a felismerő kapott hanginformációt, míg a piros szín arra utal, hogy a rendszer -kellő hangerő hiányában- nem dolgozik.
A gombok jelentése a következő:

- *Lelet indul, Lelet vége: ezzel a gombbal lehet elkezdeni, illetve befejezni a lelet diktálását.*
- *Mentés: a szövegmezőben lévő szöveget tetszőleges helyre elmenthetjük.*
- *Nyelvi tanítás: ha egy szöveget elmentettünk, akkor lehetőség van a benne lévő nyelvtani elemek megtanítására. Ha ezt a gombot megnyomjuk, akkor a nyelvtani adaptáló program elindul és a szükséges kérdések feltevése/megválaszolása után a rendszert frissíti. (Figyelem! Az új nyelvtan használatához ki kell lépni a diktálási részből és újból kezdeni a mikrofonbeállítástól.)*
- *Vágólapra: a szövegmezőben lévő szöveg a vágólapra kerül, így az a CTRL+C, vagy a Copy/Paste segítségével bármelyik dokumentumba beilleszthető.*

A programból bármikor kiléphetünk azt Alt+F4 billentyűkombináció, vagy a bezárógomb segítségével.



1.12. 6. A programról – fejlesztőknek

A szoftver kódja mindenhol dokumentálva van a kellő mértékben, hogy a Markov-modellek mély szintű megismerése után a programozó bármilyen fejlesztésbe kezdhessen. Kiemelendő azonban, hogy a szükséges matematikai, statisztikai, programozási ismeretek és a szoftver memóriamenedzsmentjének teljes feltérképezése nélkül semmiképpen se kezdjünk bele lényeges módosításokba. Az eredeti szoftver MD5-hash ellenőrző kóddal rendelkezik, minden olyan alkalommal, amikor ellenőrzi a szoftvert, annak forráskódját ellátja ezzel a jelöléssel. Így az illetéktelen, nem egyeztetett változtatások azonnal kiszűrhetőek.

2. 4. fejezet – Disztribúció

A szoftver el lett látva telepítőprogrammal, így egy új számítógépre való installálása nem tart tovább néhány kattintásnál. (Fejlesztői munka esetén azonban a forrásokat manuálisan kell másolni.) A telepítéskor néhány ellenőrzést is végez a program, melynek során csak az engedélyezett számítógépekre engedélyezi a telepítést. A telepítéshez szükséges minimális konfiguráció:

- *Pentium, vagy újabb processzor*
- *Windows 98 Se, Windows XP operációs rendszer. (Fejlesztéskor kizárólag XP-n történt a tesztelés, régebbi operációs rendszer esetében a helyes működés nem garantált.)*
- *több, mint 256 MB memória*
- *legalább 800x600-as képernyőfelbontás 16 bites színmélységben.*

Ezen felül a program minden futáskor ellenőrzi a számítógép aktuális sebességét. 1500 MHz alatti frekvencia esetén a szoftver egy üzenet után automatikusan bezár, ugyanis ez a sebesség nem elegendő a futtatáshoz. 1900 MHz alatt a szoftver egy figyelmeztető jelzést küld, mivel ilyen sebességnél elképzelhető, hogy a valós idejű munka érdekében a program az információ egyes részeit eldobja. (Csökkentett mód)

Ajánlott konfiguráció:

- *512 MB vagy több RAM*
- *2GHz-es, vagy gyorsabb processzor*

Disztribúció folyamán a telepítést végző személynek lehetősége van a felhasználók bejegyzésére, de ezt a felhasználók maguk is elvégezhetik az első bejelentkezéskor. Fontos tudnivaló, hogy egy névvel (azonosítóval) az adaptációk miatt csak egyféle munkát lehet végezni. Azaz, ha XY névvel Z munkára (pl. Gastroscopia) jegyeztük be magunkat, akkor W munkát (pl. Ultrahang) ugyanezzel az azonosítóval nem fogunk tudni végezni. Ilyen esetben a felhasználónak több azonosítóval kell rendelkezni, amelyeket meg tud különböztetni egymástól.

****

8.1. Az 1. konzorciumi tag mellékletei

8.1.2.A gasztroszkópiai és ultrahangleletező beszédfelismerő

Felhasználói kézikönyv