4 gyakorlat

Elkezdünk ismerkedni a HTK (Hidden Markov Model Toolkit) nevű beszédfelismerő rendszerrel:

* ez egy nyílt forráskódú, rejtett Markov-modelleken alapuló beszédfelismerő rendszer, sokáig a tudományos életben is egyfajta etalonnak számított (most már inkább a Kaldi az)
* A Cambridge-i egyetemen fejlesztették, ingyenesen netölthető a htk.eng.cam.ac.uk címről (regisztráció kelleni fog)
* alapvetően C-ben írva, és linux alá írták, de némi macerával Windows alatt is fordítható
* én a legfontosabb részeket (windows verzióban) elérhetővé fogom tenni, tehát nem fog kelleni fordítgatni
* sok kis futtatható exe-ből áll, ezek scriptekből hívhatók – esetünkben kis batch fájlokat fogok példaként rendelkezésre bocsátani
* dokumentáció: https://www.inf.u-szeged.hu/~tothl/speech/htkbook.pdf
* elsőként egy egyszerű számfelismerőt fogunk megcsinálni: https://www.inf.u-szeged.hu/~tothl/speech/Numbers-Demo.zip

A HTK fő moduljai:

* előfeldolgozás: HCopy
* HMM akusztikus modellek tanítása: HInit, HRest, HeRest
* nyelvi modell készítése: HParse, HBuild
* felismerés: HVite vagy HDecode
* kiértékelés: HResults

Egy beszédfelismerő alapvetően az alábbi komponensekből fog állni:

 - akusztikus modell (a tanító adatból tanuljuk)

 - nyelvi modell (megadható nyelvtanként vagy tanítható szöveges adaton)

 - kiejtési szótár (elkészíthető kézileg)

A HTK által használt fájlformátumok:

 - HTK szereti a szöveges fájlokat, ezekbe könnyen bele tudunk nézni

 - input jellemzők: .mfc formátum (saját bináris formátum)

 - címkefájlok: szöveges formátum

 - kiejtési szótár: szöveges formátum

 - akusztikus modell: szöveges vagy bináris formátum

 - nyelvi modell: szöveges vagy bináris formátum

Legelőször a HTK előfeldolgozási lehetőségeivel fogunk megismerkedni. Illetve még azelőtt az adatokkal és adatformátumokkal. Töltsük le és csomagoljuk ki az alábbit: <https://www.inf.u-szeged.hu/~tothl/speech/Numbers-Demo.zip>

A HTK-van vagy egyfajta könyvtárstruktúrája, ezt egyáltalán nem kötelező tartani, de a bemutatóban alapvetően követni fogom a htkbook demóiban is használt struktúrát. A zip-ben az alábbi könyvtárakat találjuk:

ann – ez nem része a HTK-nak, csinálunk majd egy kis neuronhálós kísérltet, ahhoz raktam be

hmm – ide kerülnek majd a betanított akusztikus modellek

lab – ide kerülnek a betanításhoz szükséges címkefájlok

mfc – ide kerülnek a betanításhoz szükséges jellemzőfájlok

network – ide kerül majd a nyelvi modell, illetve a nyelvi modellhez kapcsolódó fájlok

orts – itt lesznek a szószintű címkefájlok

A legelső kísérletekben egy ősrégi magyar nyelvű adatbázison fogunk tanítani, amelyben a beszélők a magyar számokat mondák fel, egész pontosabban olyan 26 szólakot, amelyekből bármely magyar szám összerakható 0-tól 999 999 999-ig (nulla, egy, kettő,…, tíz, tizen…). Minden hangfájl (wav fájl) egyetlen ilyen szót tartalmaz. Maguk a hangfájlok nincsenek a zipben, mert túl nagy helyet foglalnának el. Összesen 2185 ilyen fájlon fogunk tanítani és 1247fájlon tesztelni.

A fájlokban összesen 29 féle fonetikai címke fordul elő, ezek listája a hmm/hmmlist nevű fájlban található meg (szövegfájl). A fonetikai kódrendszer SAMPA-n alapul, de nem teljesen az…

Minden egyes fájlhoz kétféle címkézés áll rendelkezésre: egy beszédhang-szintű és egy szószintű. A beszédhang-szintű címkék a lab könyvtárban vannak, ismerkedjünk meg a formátumukkal (pl. az 1.lab megnyitásával)!

0 1320000 si

1320000 2640000 E

2640000 3340000 z

3340000 5060000 E

5060000 5930000 r

5930000 7230000 si

Látható, hogy esetükben a fonetikai címkékhez szegmentálási infó, azaz kezdő- és végidőpont is rendelkezésre áll. Az időt a HTK sajátos formátumban tárolja: 4 nulla levágásával kapunk milliszekundumot (pl. 1320000=132msec=0.132 sec)

A fájlokhoz szószintű címke is rendelkezésre áll, illetve pontosabban csak a tesztfájlokhoz. Ez azért kell, mert a tanítás során fonetikai szintű modelleket fogunk tanítani, de a felismeréskor a végső célunk szavak felismerése lesz, nem beszédhangok felismerése. Nézzünk bele az orts/test/1.ort fájlba!

A beszédhangok és a szavak közti kapcsolatot a kiejtési szótár teremti meg. Ezt is nézzük meg: network/dict26 fájl (ismét sima szövegfájl)

A HTK-ban előfeldolgozást (jellemzőkinyerést) a HCopy.exe nevű modullal végezhetünk (azért van ilyen furcsa neve, mert fájlok összefűzésére és egyéb manipulációkra is alkalmas). A htkbook-ban a 70-80. oldalakon rövid ismertetést olvashatunk az előfeldolgozási eljárásokról. A támogatott módszerekről pedig egy összefoglaló táblázatot találhatunk hátul, a HCopy parancs ismertetésénél (259. o):



Az előfeldolgozás típusát és paramétereit egy szöveges config fájlban kell megadni. Például az előadáson bemutatott, legnépszerűbb 13 MFCC+Δ+ΔΔ jellemzőkészlet kinyeréséhez a preprocess.config fájlt használhatjuk.

A HCopy legegyszerűbb paraméterezése:

* HCopy.exe -C config input.wav output.mfc

A jellemzőfájlok tárolásához a HTK-nak van egy saját bináris fájlformátuma (egy pár bájtos headert pakol az adatok elé). A kinyert jellemzőfájlokat az mfc könyvtárba fogjuk pakolni.

Általában a betanításhoz hangfájlok százai vagy ezrei kellenek. Az input-output párokat egy szövegfájlban is felsorolhatjuk, és az –S filelist kapcsolóval adhatjuk át a HCopy-nak. Példa: trainlist.hcp, testlist.hcp

Esetünkben az előfeldolgozást már előre elvégeztük a runhcopy.bat fájllal.

Megjegyzés: a –T kapcsolóval lehet állítani, hogy milyen részletes outputot (debug infót) kapjunk a képernyőn a modulok futása közben

Neuronhálós kísérlet

A tanítás inputját tehát az mfc (jellemzővektorok) és lab (címkefájlok) párosok fogják adni. Az összetartozó mfc és lab fájlokat az azonos fájlnév alapján fogja tudni összepárosítani a rendszer.

Most elvégzünk egy egyszerű kísérletet, melyben megnézzük, hogy az egyes jellemző-vektorok milyen pontossággal ismerhetők fel a lab fájlokban hozzájuk rendelt beszédhang-ként. Ehhez egy régi neuronháló-implementációt fogunk használni, ez nem része a HTK-nak, csak érdekességképp nézzük meg.

Ehhez először át kell konvertálnunk az inputot a neuronháló által elvárt formátumra, ami egy egyszerű szöveges formátum, soraiban egy-egy jellemzővektorral, majd a vektorhoz tartozó címkével. Mivel a jellemzővektorok lépésköze ismert (10ms), a lab fájlok alapján nyilván ki lehet számolgatni, hogy az mfc fájl egyes vektoraihoz konkrétan milyen címke tartozik. E konverziót az alábbi tool fogja elvégezni (szintén nem a HTK része):

pfilelabseg.exe trainlist.lab mfc/train ann/numbers.data hmm/hmmlist c 0 0 0

pfilelabseg.exe testlist.lab mfc/test ann/numbers.test hmm/hmmlist c 0 0 0

Ereményképp előáll a numbers.data tömb (231328 vektorral), valamint a number.test tömb (134767 vektor) az ANN könyvtárban – ott folytatjuk a munkát

A neuronháló tanítása előtt az jellemzővektorokat normalizálni (standardizálni) érdemes, ezt végzi el az alábbi parancs:

Normalize.exe numbers numbers-n

Végezetül, a neuronháló tanítása:

trainannNeigh.exe numbers-n 500 2 0.8-0.01-0.5 0 1 0 256

(500 rejtett neuronunk lesz, az adatok 80%-án tanítunk, a 20%-on keresztvalidálunk, a learning rate kezdőértéke 0.01 majd felezgetjük, a batch mérete 256, a kimenőrétegben softmax neuronok vannak).

a betanított háló kiértékelése a tesztadatokon:

TestANNBatch.exe numbers-n.test numbers-n.wts

82%-os pontosságot kapunk, ami elég jó. „Valódi” adatokon (spontán beszéd, minden hang előfordul,…) inkább csak 50% körüli értékek szoktak kijönni.

A háló lehetővé teszi szomszédos vektorok bevonását is az inputba. Ha például 4-4 szomszédos vektort is figyelembe akarunk venni, akkor akkor az utolsó 0 paramétert 4-re kell átírni (és a wts súlyfájlt újratanítás előtt le kell törölni!). Ismételjük meg így is a tanítást! Figyeljük meg, hogy 39 helyett mekkora lett az input réteg mérete! Mennyit javult a felismerés pontossága?

5. gyakorlat

Az előző órán megnéztük, hogyan lehet elvégezni az előfeldolgozást (jellemzőkinyerést) az adatokon, a kinyert jellemzők az mfc/train és mfc/test könyvtárakba kerültek. A tanításhoz a jellemzőkön kívül még egy dolog fog kelleni, a tanítócímkék. Esetünkben rendelkezésre áll fonetikai szintű címkézés és szegmentálás is (a lab könyvtárban). Ebben az esetben egyszerűbb a tanítás, mivel minden beszédhangnak ismert a címkéje és a helye is. Először azt nézzük meg, hogy ilyen esetben hogyan tudunk tanítani. Következő órákon fogjuk megnézni azt az esetet, amikor fonetikai címkézés és szegmentálás nem áll rendelkezésre, csupán szószintű átiratok.

A tanítás során minden egyes beszédhanghoz betanítunk egy-egy HMM-et. A betanított modellek a hmm könyvtárba fognak kerülni. A modellek tanítása előtt létre kell hoznunk egy „üres” modellt, egy prototípust, ebben definiáljuk a modell struktúráját. A tanítás során ez az üres modell töltődik fel a modell tanult paramétereivel.

Egy ilyen prototípust találunk a hmm/proto3state3gauss fájlban (szövegfájl, meg lehet nyitni). Egy HMM kétféle komponensből fog állni: állapotokból és eloszlásokból. Az eloszlások leírására GMM-et használ, azaz Gauss-görbék súlyozott összegét. A „NumMixes” paraméter tárolja a Gauss-görbék számát, a Mean és Variance paraméterek az átlagot és a szórást minden egyes görbéhez. Jelen példában az állapotok száma 3 lesz, ezekhez tartozik egy állapotátmenet-mátrix (TansP). A HTK implementációjában minden modellhez tartozni fog egy első és egy utolsó állapot, amelyhez nincs valószínűségi eloszlás, ezek csak a modellek összefűzésére vannak ott. Ezért látunk a fájlban 5 állapotot 3 helyett.

Amennyiben rendelkezésre áll fonetikai címkézés és szegmentálás (mint esetünkben), akkor a modelleket a HInit.exe paranccsal lehet inicializálni, majd a HRest.exe paranccsal tovább tanítani (Baum-Welch algoritmus, ld. órán). Az egyik lépésre a runhinit.bat, a másikat a runhrest.bat batch-fájlt készítettem elő.

A Hinit.exe paraméterezése:

-S trainlist.mfc: a tanító jellemzőfájlok listája

-L lab\train: a hozzájuk tartozó lab fájlok útvonala (a fájlnév alapján párosítja az mfc-vel)

-l A: az ilyen címkéjű szegmentumokat fogja kigyűjteni és azokon tanul

-M hmm\hmm0: ebbe a könyvtárba kerül a betanított modell

-o A: ez lesz a modell neve (jelen esetben megegyezik a beszédhang címkéjével)

hmm\proto3state3gauss: a modellhez tartozó prototípus-fájl

Indítsuk el a tanítást! Megfigyelhetjük, hogy a tanulás iteratív folyamat, a lépések után a modell egyre jobban illeszkedik az adatokra (nő a log-probability értéke). Nézzük meg, hogy hogyan változtak a hmm/hmm0 könyvtárban levő modellek a prototípushoz képest!

A HRest.exe paraméterezése:

Lényegében megegyezik A Hinit.exe paraméterezésével, két különbséggel:

- Most már nem az üres prototípusból indulunk ki, hanem a hmm/hmm0 modelljeiből

- A finomított modelleket a hmm/hmm1 könytárba tesszük

A betanítás után jöhet a felismerés! Ehhez a tesztadatokon és a betanított modelleket kívül még két dolog fog kelleni:

Kiejtési szótár: ez ezért kell, mert beszédhang-modelleket tanítottunk, de a felismerés kimenetén nem beszédhangokat, hanem szavakat vagy szó-sorozatokat szeretnénk kapni. A hangok és a szavak közti kapcsolatot a kiejtési szótár teremti meg (network/dict26, korábban már megnéztük).

Nyelvi modell: bármilyen egyszerű felismerési kísérletet is szeretnénk végezni, mindenképpen kelleni fogy egy nyelvi modell. A nyelvi modellnek a HTK-ban két fő típusa lehet:

1. Nyelvtannal megadott nyelvi modell. Ilyenkor szabályok segítségével kell megadni, hogy mely szavak és hogyan követhetik egymást. E szabályok megadásához a HTK-nak van egy saját leíró rendszere, ez nagyon hasonlít ahhoz, mintha reguláris kifejezéseket adnánk meg, de automatához is hasonlíthatom.
2. Sztochasztikus nyelvi modell. Ez az jelenti, hogy a szavak bizonyos valószínűséggel követik egymást. Az ilyen nyelvi modelleket tanítással fogjuk kialakítani, ezekről az előadáson majd később lesz szó.

Mi most nyelvtanos modellt fogunk használni, mégpedig a lehető legegyszerűbbet. Ugyanis tudjuk, hogy minden tesztfájlban pontosan egy szó szerepel, tehát egyetlen szót kell felismerni. Ilyen ún. izolált szavas felismerés esetén a nyelvtan így néz ki:



A nyelvtanunk egy szövegfájl megadásával tudjuk definiálni, esetünkben a nyelvtan a number/numberNetwork fájlban van megadva.

(: kezdőállapot

): végállapot

|: vagy művelet

Ez a HTK által „network”-nek nevezett formátum. A használathoz át kell fordítanunk egy belső, ún. „lattice” formátumba a HParse.exe segítségével:

HParse.exe numberNetwork numberLattice

Most már tényleg elindíthatjuk a felismerést. Erre két modult lehet használni a HTK-ban, az egyik a HVite.exe, a másik a HDEcode.exe. A paraméterezésük nagyon hasonló, egyszerűbb feladatok esetén kényelmesebb a HVite, a HDecode inkább nagyobb feladatoknál (folyamatos, nagyszótáras felismerés) javasolt. Ezért mi most a HVite-t fogjuk használni, ld. runhvite.bat

-S testlist.mfc: tesztfájlok listája

-d hmm\hmm1: akusztikus modellek helye

-w network\numberlattice: nyelvi modell

-i results.mlf: ide kerül az eredmény

network\dict26: kiejtési szótár

hmm\hmmlist: akusztikus modellek listája

Futtassuk le a felismerést!

A képernyőre is írja az eredményt, de a results.mlf fájlban is megnézhetjük, mit adott ki: minden fájlhoz megkapjuk a legvalószínűbb szót, annak log-valószínűségét, a kezdő- és záró időpontját.

Minden fájlhoz kaptunk tehát egy eredményt, de egyelőre nem tudjuk, hogy az jó-e. Ennek az eldöntéséhez a kapott eredményeket az otr/test könyvtárban levő szószintű címkékkel kell összehasonlítani. Ezt persze nem kézzel kell megtennünk, erre szolgál a HResults.exe modul, illetv a hunhresults.bat fájl. Paraméterek:

-L orts\test: az összehasonlítás alapját adó címkefájlok helye

-X ort: a címkefájlok kiterjesztése (mert eltértünk a default lab kiterjesztéstől)

network\wordlist: a lehetséges szavak listája

results.mlf: az összehasonlítandó címkék

Eredményként mondatszínű (SENT) és szószintű (WORD) értékeket is kapunk. Esetünkben mindhárom szám ugyanaz, mivel egy fájlban („mondat”) pontosan egy szó van. Később megbeszéljük, hogy hogyan végezzük a kiértékelést többszavas kimenetek (mondatok) esetén.

99.44%-os eredményt kaptunk. Próbáljuk ki ugyanezt a hmm0 könyvtárban lévő modellekkel is!

6. gyakorlat

Próbáljuk ki, hogy mi történik, ha egy olyan szót próbálunk felismerni, amelyet a felismerő nem ismer! A fájlok között található a „kőszikla.wav” nevű fájl, ez lesz az inputunk. Ebben a „kőszikla” szó hangzik el, ami nyilvánvalóan nem egy szám, a felismerő szótárában nem szerepel. Ha be akarjuk engedni a felismerőbe, akkor nyilván el kell rajta végezni az előfeldolgozást:

HCopy.exe –C preprocess.config koszikla.wav koszikla.mfc

A konkrét felismeréshez a hvite.bat-on annyi módosítást kell végeznünk, hogy most nem a –S testlist.mfc fájllista lesz a bemenet, hanem egyetlen fájl. Ezt átadhatjuk az alábbi módon:

HVite -T 1 -C nobyteswap.config -d hmm\hmm1 -w network\numberlattice -i results.mlf network\dict26 hmm\hmmlist koszikla.mfc

Eredményként a „huszon” szót kaptuk. A felismerőnek esélye sem volt felismerni, hiszen az a szó nem szerepel a szótárában. Az ilyen szavakat out-of-vocabulary, vagy röviden OOV szavaknak hívjuk, és nyilvánvaló, hogy garantáltan hibaként fognak jelentkezni a felismerő kimenetén. Egy egyszerű dolgot lehetne tenni védekezésként: a szóhoz rendelt valószínűség alapján lehet sejteni, hogy a rendszer bizonytalan az eredményben. Be lehetne lőni egy küszöböt, ami alapján a rendszer bizonytalan esetekben inkább nem ad kimenetet.

Mit kell tennünk, ha ezt a szót is be szeretnénk venni a felismerőbe? Mivel az összes, a szóban szereplő hanghoz van betanított akusztikus modellünk, ezért az akusztikus modellt NEM kell módosítanunk (ez az egyik előnye annak, hogy beszédhangokból építkezünk!). A kiejtési szótárt és a nyelvi modellt viszont módosítanunk kell:

A kiejtési szótárhoz (network/dict26) adjuk hozzá az alábbi sort:

kőszikla si k qq s i cl k l A si

A nyelvi modellt, azaz a numberNetwork fájlt is ki kell bővítenünk az alábbival:

| kőszikla

Ne fejeltsük el, hogy ezt újra is kell „fordítani” lattice fájllá:

HParse.exe –T 1 numberNetwork numberLattice

Futtassuk le újra a HVite parancsot! Látjuk, hogy most már felismeri a rendszer a szót.

Bár a tanító adatbázisban csak 26 szó szerepel, ezek úgy lettek összeválogatva, hogy elvileg minden számot össze lehessen belőlük rakni 0-tól 999 999 999-ig. Ehhez a nyelvtan kell módosítanunk. Elsőként próbálkozzunk meg azzal, hogy megengedjük a szavak ismétlődését. Ehhez a nyelvtant így kell módosítanunk (balra a jelenlegi, jobbra az új):



(Ezzel tulajdonképpen átlépünk izolált szavas felismerésből folyamatos beszéd felismerésébe!)

A visszacsatoló hurokél megengedéséhez a numberNetwork fájlunkban egyetlen apró módosítás szükséges: a szavainkat < > jelek közé kell tennünk

(<

egy | kettő | három | négy | öt | hat | hét | nyolc | kilenc | tíz |

tizen | huszon | húsz | harminc | negyven | ötven | hatvan | hetven |

nyolcvan | kilencven | száz | kettőszáz | ezer | kettőezer | egymillió | nulla

>)

(És persze ne felejtsük el a HParse-vel ismét „újrafordítani”.)

A teszteléshez két hosszú számot raktam be a fájlok közé: 314296.wav, illetve 906182.wav

Persze ezeket is át kell engedni az előfeldolgozáson:

HCopy.exe -C preprocess.config 314296.wav 314296.mfc

Majd mehet a felismerés:

HVite -T 1 -C nobyteswap.config -d hmm\hmm1 -w network\numberlattice -i results.mlf network\dict26 hmm\hmmlist 314296.mfc

A kapott eredmény nem lett valami fényes: harminc tizen tizen kettőszáz hatvan

Mit rontottunk el?

A kiejtési szótárban az egyes szavak fonetikai átirata így néz ki pl:

ezer si E z E r si

Azaz minden szónak csenddel kell kezdődnie és csenddel kell végződnie (si). Azonban amikor a 314296-ot kimondjuk, akkor nem tartunk szünetet az építőelemek között.

Tehát a nyelvtant inkább így kellett volna átalakítani:



Ehhez a csendet (sil) külön „szóként” fogjuk kezelni, a Network fájlt pedig így kell módosítani (én numberNetworkLoop néven mentettem el):

(sil

<

egy | kettő | három | négy | öt | hat | hét | nyolc | kilenc | tíz |

tizen | huszon | húsz | harminc | negyven | ötven | hatvan | hetven |

nyolcvan | kilencven | száz | kettőszáz | ezer | kettőezer | egymillió | nulla

>

sil)

Ismét ne feledkezzünk el a HParse alkalmazásáról!

Emellett persze a kiejtési szótárat is át kell alakítani, a csendet kiemelve külön „szónak”. A megfelelő kiejtési szótár megtalálható network/dictConnected alatt.

És végre a felismerés:

HVite -T 1 -C nobyteswap.config -d hmm\hmm1 -w network\numberlatticeloop -i results.mlf network\dictConnected hmm\hmmlist 314296.mfc

Még egy dolgot tudunk pontosítani a nyelvtanon: a jelenlegi „körbehurkolt” nyelvtan megenged olyan szósorozatokat, mint pl. ezerezerezer, pedig ez nem szabályos szám. Ha csak helyes számalakokat akarunk elfogadni, akkor egy jóval bonyolultabb nyelvtant kell összeraknunk, erre ad példát a connectedNetwork nevű nyelvtanfájl. Láthatjuk, hogy bonyolultabb nyelvtanokban részkifejezések leírására tudunk változókat is felvenni a $ jel segítségével.

Ezzel a nyelvtannal így tudunk felismerni:

HVite -T 1 -C nobyteswap.config -d hmm\hmm1 -w network\connectedLattice -i results.mlf network\dictConnected hmm\hmmlist 314296.mfc

Mit tudunk csinálni, ha arra lennénk kíváncsiak, hogy mennyire működne jól a felismerőnk, ha csak az akusztikus modellekre támaszkodhatna, nyelvi segítség nélkül? Ennek megvizsgálására szószintű helyett fonetikai szintű felismerést szeretnénk végezni, amikor is bármilyen hangsorozat megengedett a kimenetben, azaz nincs támogatás se a kiejtési szótár, se a nyelvtan részéről.

A HTK sajnos minden estben elvárja, hogy legyen kiejtési szótár és nyelvi modell, de be tudjuk csapni: csináljuk egy olyan szótárat, amelyikben minden hangnak saját maga felel meg szóként, és egy olyan nyelvi modellt, amelyikben bármelyik hang követheti bármelyik hangot. Ezekre található megoldás a network/dictphone, illetve network/phoneLattice fájlokban. A felismerőt most az összes teszfájlon le fogjuk futtatni, ebben segít a runhvitep.bat

A kiértékelést pedig a runhresultsp.bat paranccsal végezhetjük el. Ezt kapjuk:

------------------------ Overall Results --------------------------

SENT: %Correct=23.26 [H=290, S=957, N=1247]

WORD: %Corr=92.99, Acc=77.21 [H=8593, D=99, S=549, I=1458, N=9241]

Értelmezés:

A SENT sor azt mutatja, hogy hány esetben találta a teljes fájl tartalmát (SENTENCE).

A WORD sor azt mutatja, hogy hány százalékban találta el az egyes szavakat – esetünkben ezek most a hangok.

A felismerő kimenete egy hangsorozat (R), és a helyes átirat is egy hangsorozat (C). Hogyan tudjuk ezeket összehasonlítani? Példa:

R: á l m a

C: a l m a

Láhatjuk, hogy ebben az esetben egyetlen hiba van, egy hangcsere (substitution). De mi a helyzet pl. ilyenkor:

R: a l a

C: a l m a

Vagy ilyenkor:

R: a l a m a

C: a l m a

Az előbbi esetben törlési (deletion), az utóbbi esetben beszúrási (insertion) hibáról beszélünk. Hosszabb sorozatok, illetve sok hiba esetén nem triviális, hogy pontosan melyik hanggal mi történt. Ilyenkor jön a képbe a szerkesztési távolság (edit distance,  [Levenshtein distance](https://en.wikipedia.org/wiki/Levenshtein_distance%22%20%5Co%20%22)):

A három műveletet használva (substitution, deletion, insertion) próbáljuk meg áttraszformálni egyik stringet a másikba. Sajnos ez nem csak egyféle módon tehető meg hiszen pl. egy csere ekvivalens egy törlés-beszúrás párral. Vizsgáljuk meg az összes lehetséges megoldást, és definiáljuk a távolságukat a lehető legkevesebb művelettel járó transzformáció műveletszáma (bonyolultabb esetben lehet költségeket is rendelni az egyes műveletekhez)!

A HResults parancs az edit distance alapján számolja ki a D, S, I műveletek számát.

Egyetlen mérőszámot, pl az accuracy értéket pedig így kaphatunk ezekből:

A = (N-D-S-I)/N, ahol N az összes hang száma a helyes átiratban.

(A Correctness érték ugyanez, de az insertion-öket nem veszi figyelembe).

Bár most beszédhangokkal mutattam be, folyamatos beszéd esetén szósorozatokra is ugyanezt a kiértékelést szokták elvégezni.