PROGRAMSKI PAKET AlfaNumCASR
sistem za prepoznavanje kontinualnog govora

Darko Pekar, Radovan Obradović, Vlado Delić

FTN, Trg Dositeja Obradovića 6, 21000 Novi Sad
e-mail: pekard@eunet.yu

SADRŽAJ

U radu je predstavljen programski paket za prepoznavanje kontinualnog govora koji za sada uspešno funkcioniše na malim i srednjim rečnicima. Predstavljeni su moduli za obuku i prepoznavanje, a na kraju je dat i kratak pregled modula koji su još uvek u fazi ispitivanja.

1. UVOD

Programski paket AlfaNumCASR proizvod je višegodišnjeg rada autora na problematici prepoznavanja govora, počevši od prepoznavanja izolovanih reči, preko povezanih reči [2], pa sve do prepoznavanja kontinualnog govora. S obzirom da je baziran na prepoznavanju fonema u kontekstu sistem podržava zadavanje proizvoljnog skupa reči (gramatike) koje treba prepoznavati. Programski paket je pisan u programskom jeziku C++ i u potpunosti je razvijen od strane autora. Pisan je tako da je u najvećem delu nezavisan od platforme i operativnog sistema (osim u delu gde se zahteva komunikacija sa hardverom). U paket su uključene i dve ranije razvijene biblioteke [1]. To su **slib** biblioteka za obradu signala i **an_misc** biblioteka opšteg tipa.

AlfaNumCASR paket podeljen je u nekoliko celina. Objašnjenje će biti dato kroz faze obuke i testiranja u procesu kreiranja jednog ASR sistema. Te faze obuhvataju:

- formiranje i/ili podešavanje globalnog konfiguracionog (**ini**) fajla,
- izdvajanje obeležja,
- formiranje label fajlova,
- inicijalna obuka,
- ML trening,
- gramatika i njeno zadavanje,
- prepoznavanje.

2. OPŠTE INFORMACIJE

2.1 Konfiguracioni fajl

U ovom tekstualnom fajlu se zadaju svi parametri koji su važni kako za obuku tako i za prepoznavanje. Ovo nije klasičan **ini** fajl već ima malo drugačiju strukturu. Podržane su liste i vektori, tako da se u ovom fajlu mogu opisivati proizvoljno složene strukture. Ovo je omogućeno postojanjem klase *sobj* kao i odgovarajućeg parsera. Evo kratkog primera kako mogu izgledati podaci u **ini** fajlu:

```
{
// features data
   samples_per_second = 16000.0,
   window_duration_ms = 30.0,
   frame_duration_ms = 10.0,
   derivatives_data =
   [
      { window_ms = 30.0, weight = 2.0 }
      { window_ms = 50.0, weight = 2.0 }
   ]
   cache_dynamic_features = 0 // false
}
```

Inače pri konfigurisanju parametara obuke i testiranja postoje tri nivoa. Prvi je *default* nivo, odnosno podrazumevane vrednosti parametara. Ove vrednosti su zadate u samom kodu i ispisuju se svaki put kad se startuje program. Drugi nivo je konfiguracioni fajl, u kojem se može premostiti bilo koja *default* vrednost. Treći nivo je komandna linija u kojoj možemo zadati nove vrednosti kojima se premošćavaju one iz prethodnih nivoa. Na primer ako u komandnoj liniji piše:

```
csr test -cdf 1
```

switch **-cdf** omogućava da se postavi vrednost promenljive **cache_dynamic_features** na 1, mada je u konfiguracionom fajlu bila 0. Međutim nemaju sve promenljive mogućnost da se menjaju iz komandne linije (nemaju obezbeđene *switch*-eve), već samo one za koje to ima smisla.

2.2 Izdvajanje obeležja

Izdvajanje obeležja bazira se na biblioteci **slib** [1] i njenoj script verziji **slib_script**. Naime ekstraktor obeležja zadaje se kao tekstualni fajl u kome su opisani blokovi koji se koriste za računanje obeležja. Ovde korisnik ima na raspolaganju širok spektar najčešće korišćenih obeležja kao što su: kepstralni koeficijenti (nekoliko vrsta), energija i log energija, broj preseka sa nulom (*zero crossing rate*), osnovna učestanost, stepen zvučnosti, kao i izvodi ovih statičkih obeležja. Dakle obeležja nisu fiksirana u kodu, već korisnik ima mogućnost da sam izabere obeležja koja želi. Obzirom da je kreiranje kompletnog fajla za ekstrakciju obeležja relativno složeno, ponuđeno je nekoliko najčešće korišćenih kombinacija. Korisnik može koristiti ponuđene ili ih modifikovati po želji.

Program podržava tzv. keširanje obeležja na disk. Naime poznato je da proces izdvajanja obeležja zahteva prilično vreme, a kada jednom ustanovimo obeležja obično ih ne menjamo tokom obuke i testiranja. Zbog toga je zgodno izdvojena obeležja snimiti na disk i u naknadnom krugu obuke ih praktično trenutno učitati. Ovaj proces se odvija potpuno automatski, s tim da ga možete isključiti. Naime, na osnovu imena i sadržaja ekstraktora koji se koristi, kreira se odgovarajući

direktorijum u koji se snimaju obeležja dobijena iz pojedinih "wav" fajlova. Pri tome je struktura direktorijuma u koju su smešteni ovi fajlovi ista kao ona u kojoj su se nalazili početni "wav" fajlovi. Naglašavamo da se ime ovog direktorijuma određuje ne samo na osnovu imena korišćenog ekstraktora, već i na osnovu njegovog sadržaja (računa se *hash* vrednost sadržaja fajla), kao i još nekih parametara koji utiču na izgled obeležja, tako da se ne može desiti da program učitava obeležja sa diska, a mi smo nešto promenili u načinu njihovog računanja. Svi ovi fajlovi smeštaju se u direktorijum "cache", tako da se nepotrebni fajlovi mogu ručno obrisati.

Zanimljiv podatak je da se obeležja snimaju na disk u formatu koji je kompatibilan sa HTK parametar fajlovima [4]. Ovo je ovako urađeno iz dva razloga. Da bi se proverilo kako naša obeležja funkcionišu na sistemu koji je obučio HTK, i da bi utvrdili prednosti ili mane našeg sistema za obuku u poređenju sa HTK.

Delta obeležja se mogu računati na dva načina. Prvi pristup je računanje direktno u ekstraktoru, pomoću **slib** blokova, što je pristup koji se preporučuje ako želimo sistem da koristimo za neko *on-line* prepoznavanje (odnosno slučaj kada nemamo na raspolaganju ceo fajl, već odbirci sukcesivno dolaze). Drugi pristup je naknadno računanje u programu, kada se u **ini** fajlu definišu broj izvoda koji se koristi, kao i prozori pojedinih izvoda. Ovaj pristup može da nam uštedi prostor na disku prilikom keširanja, jer se snimaju tri puta manji fajlovi sa obeležjima (ako koristimo dva izvoda). U drugom slučaju izvodi se pri učitavanju keširanih fajlova računaju "u letu", što ne uzima puno vremena.

Normalizacija energije takođe može da se obavi na dva načina, slično kao i delta obeležja. Ukoliko nam treba *on-line* prepoznavač koristimo **slib** blokove, a normalizacija se obavlja u tzv. pomerajućem prozoru (**SMovingMax** i **SMovingMin** blokovi) uz izvesno, konfigurabilno, kašnjenje. Kad se radi obrada iz fajla, mnogo bolji način je normalizacija signala na nivou celog fajla. Ovo se takođe specificira u **ini** fajlu.

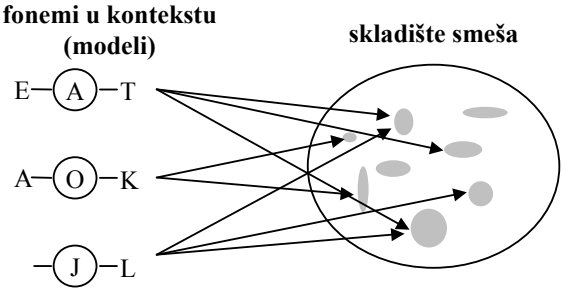
Podržana je i poznata metoda za izdvajanje robusnijih kepstralnih koeficijenata, otpornih na promene kanala, *cepstral mean subtraction* [6]. Ona se takođe može sprovoditi "u letu" ili na nivou fajla.

Obeležja se, nakon izdvajanja, mogu podeliti u nekoliko strimova (*data stream*). Koja će obeležja pripadati kojem strimu i koliko će ih biti, potpuno je proizvoljno i zadaje se u **ini** fajlu. Iako je daleko najzastupljeniji slučaj kada se koristi samo jedan strim, ponekad se dobija bolja pokrivenost akustičkog prostora rastavljanjem vektora obeležja na nekolioko nezavisnih delova, naročito ako radimo sa vezanim smešama (videti niže) ili koristimo vektorsku kvantizaciju.

U strukturi u kojoj zadajemo informaciju o strimovima zadaju se i težine pojedinih obeležja. Ovo je veoma važno u inicijalnoj obuci i vektorskoj kvantizaciji, ali kad se jednom izračunaju varijanse, ova informacija postaje nebitna.

2.3 Struktura modela

Naš sistem koristi tzv. *semi continuous HMM* [6] pristup, koji znači sledeće: modeli pojedinih fonema u kontekstu nemaju svoje, i samo svoje, smeše, već postoji skladište svih smeša koje su nastale obukom, a svaki model poseduje listu indeksa onih smeša koje mu pripadaju. Ovaj pristup obezbeđuje veliku vremesku uštedu u procesu prepoznavanja, jer se značajno smanjuje broj smeša koje treba izračunati (već u zavisnosti od gramatike).



Slika 2.1: Organizacija modela

Mada se smeše dele između modela, ipak svaki model ima sopstvene težine za svaku od smeša koju koristi. Pored toga za svaki model se čuva raspodela njegovog trajanja, koja je bila prisutna u bazi za obuku. Na osnovu ovog podatka se modeluje njegovo trajanje pri prepoznavanju. Dalje, modeli sadrže oznaku fonema koji reprezentuju, kao i opcione leve i desne kontekste. Ukoliko model ima levi kontekst, onda se on može koristiti samo u tom kontekstu, odnosno samo posle tog fonema. Isto važi i za desni kontekst. Dakle ako ne sadrži ni levi ni desni, onda je to tzv. monofon koji se može koristiti u bilo kom kontekstu. Naravno, pri sastavljanju trelisa za prepoznavanje uvek se uzima najbolj model koji postoji, odnosno onaj sa najširim kontekstom koji se uklapa u traženi.

Do sada se, zbog razumljivosti, sve govorilo na nivou fonema, mada se zapravo ovde radi o subfonemima. Umesto termina subfonem, može se koristiti termin stanje, pošto su ekvivalentni. Na primer fonem "A" se može sastojati iz tri subfonema "A0", "A1" i "A2". Za svaki subfonem se pravi zaseban model, sa sopstvenim kontekstima. Ovo omogućuje da se, u nedostatku dovoljnog broja observacija u bazi za obuku, npr. za leve subfoneme ("A0") prave samo levi konteksti, a za desne samo desni. Obično se i za srednje prave desni konteksti, pošto efekti nazalizacije i jednačenja po zvučnosti idu unazad, a ne unapred. Ukoliko postoji dovoljno observacija praviće se trifoni. Broj subfonema za svaki fonem definiše se u konfiguracionom fajlu i ne mora biti isti za sve foneme.

2.4 Label fajlovi

U proceduri obuke potrebno je, pored zvučnih fajlova sa snimljenim sekvencama, obezbediti i fajl koji sadrži informacije o tome šta je u tom zvučnom fajlu rečeno i gde su granice između govornih jedinica. Pošto ovaj fajl sadrži informacije o labelama, uobičajeni naziv je label fajl. Jedna linija u našem label fajlu je sledećeg oblika:

```
"pera.wav" "gender=male;" "_ Po Pe E R A spk _ #" 0.0 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Vide se četiri celine:

1. Pod navodnicima se navodi ime "wav" fajla sa izgovorenom sekvencom.
2. **Class** polje. U njemu možemo navesti proizvoljan broj podataka tipa *key=value*. Može se koristiti za podatke o govorniku, o samom fajlu (kvalitet i sl.), o tome da li je fajl pregledan ili ne itd.
3. Labele, odnosno izgovoreni fonemi. Program ne ograničava da ovo budu strogo fonemi. Kao što vidimo ploziv "P" je razdvojen na okluziju i eksploziju. Isto tako bismo mogli čitavu reč staviti pod jednu labelu. Rezervisane oznake su jedino "_", što je oznaka za tišinu i "#" što je oznaka za završetak poslednjeg fonema (tj. labele). Vidi se i oznaka "spk" koja može da obeležava šum, poreklom od spikera, ali je to u potpunosti do korisnika programa da definiše i koristi takve oznake. Još jedno ograničenje kojeg se treba pridržavati je da se labela ne završava brojem (npr. A1), jer se ove oznake koriste interno za podelu fonema na subfoneme.
4. Granice, tačnije, počeci pojedinih fonema, izraženo u sekundama.

Broj redova u jednom label fajlu, označava broj zvučnih fajlova koji će biti učitani u procesu obuke.

3. OBUKA MODELA

3.1 Inicijalna obuka

Kao što je već rečeno, koristi se *semi continuous HMM* model, odnosno pristup sa vezanim smešama. Istu smešu može koristiti, u opštem slučaju, bilo koji model, a koji će modeli zapravo deliti smeše, zavisi od inicijalne obuke. Nema mnogo smisla dozvoliti da npr. neki frikativ i neki vokal dele istu smešu. Zbog toga se definišu tzv. grupe fonema unutar kojih modeli dele smeše. Svaka grupa je disjunktna sa drugim grupama. Ove grupe se mogu definisati u **ini** fajlu. Postoji i drugi pristup po kojem svaki subfonem predstavlja različitu grupu fonema. Npr. sve sekvence koje reprezentuju "A0" će biti smeštene u jednu grupu, a pojedini modeli (tj. "A0" u različitim kontekstima) će koristiti odgovarajuće smeše iz tog skladišta. Iako se ovim drugim pristupom pravi nešto više smeša, pokazao se mnogo uspešnijim u pogledu tačnosti prepoznavanja.

U nastavku će biti data opisna verzija algoritma inicijalne obuke.

```
main
{
    Izdvoj obeležja i sve kontekste raspoložive u bazi;
    Za svaku grupu fonema:
    {
        Pokupi sve vektore koji opisuju bilo koji fonem iz
        grupe;
        Vektorska kvantizacija (pravljenje skladišta
        smeša);
        Za svaki subfonem iz grupe:
        {
            Pokupi sve vektore koji opisuju taj subfonem;
            NapraviMonofon(vektori, subfonem);
        }
    }
}

NapraviMonofon(vektori, subfonem)
{
    Ako je levi subfonem (npr. A0 od A0 A1 i A2):
    {
        Za sve leve kontekste raspoložive u bazi:
        {
            Prebaci sve vektore koji se tiču tog konteksta;
            Ako je broj prebačenih vektora veći od
            definisanog za bifone i broj preostalih vektora
            veći od definisanog za monofone:
            {
                NapraviLeviBifon(prebačeni vektori, levi
                kontekst);
            }
            else
                Vrati vektore;
        }
    }
    else (srednji ili desni subfonem)
    {
        Za sve desne kontekste raspoložive u bazi:
        {
            Prebaci sve vektore koji se tiču tog konteksta;
            Ako je broj prebačenih vektora veći od
            definisanog za bifone i broj preostalih vektora
            veći od definisanog za monofone:
            {
                NapraviDesniBifon(prebačeni vektori,
                desni kontekst);
            }
            else
                Vrati vektore;
        }
    }
    NapraviInicijalniModel(preostali vektori) - monofon;
}

NapraviLeviBifon(vektori, levi kontekst)
{
    Za sve desne kontekste raspoložive u bazi:
    {
        Prebaci sve vektore koji se tiču tog konteksta;
        Ako je broj prebačenih vektora veći od definisanog
        za trifone i broj preostalih vektora veći od
        definisanog za bifone:
        {
            NapraviInicijalniModel(prebačeni vektori) -
            trifon;
        }
        else
            Vrati vektore;
    }
    NapraviInicijalniModel(preostali vektori) - bifon;
}

NapraviDesniBifon(vektori, desni kontekst)
{
    ...
```

```
}

NapraviInicijalniModel(vektori)
{
    Pokupi sve smeše u kojima se nalazi makar jedan
    vektor;
    Izračunaj prosečnu vrednost gustine verovatnoća
    (pvgv) vektora u svojim smešama;
    while (pvgv > definisane vrednosti)
    {
        Izbaci smešu sa najmanje vektora, a te vektore
        pridruži drugim, najbližim, smešama;
        Izračunaj prosečnu vrednost gustine verovatnoća
        (pvgv) vektora u svojim smešama;
    }
    Od preostalih smeša napravi model;
}
```

U poslednjem koraku kada se od preostalih smeša pravi model, inicijalne težine pojedinih smeša se postavljaju srazmerno broju vektora preostalih u toj smeši. Nije napisano u algoritmu, ali u istoj proceduri se izračunavaju i raspodele trajanja pojedinih modela. Dakle pamti se čitav grafikon njegovog trajanja u bazi za obuku, koji će se kasnije pri prepoznavanju moći koristiti na dva načina.

Čak i kada ograničimo grupe fonema na samo jedan subfonem, još uvek može doći do relativno nelogičnih podela smeša. Na primer, nema mnogo smisla da se dele smeše između modela S-A0 ("A0" sa levim kontekstom "S") i modela E-A0. Ima smisla deliti smeše između onih modela u kontekstu kod kojih je kontekst istorodan. Znači u jednu grupu bi trebali ići konteksti koje čine vokali, u drugu firkativi i sl. Ovaj problem je u HTK rešen tzv. *tree-based-clustering* algoritmom, a kod nas nije eksplicitno tretiran. Međutim ako se težine pojedinih vektora obeležja pametno podese, može se dobiti sasvim konzistentna raspodela smeša po modelima. Trebalo bi iz ovih razloga dati veliku težinu sledećim obeležjima: energija, prva dva kepstrala (znači grubo opisivanje spektra koje može veoma efikasno da razdvoji frikative, nazale i vokale) kao i njihova delta obeležja (da bi opisali kontekst tj. promenu tih obeležja).

Vidimo da je procedura inicijalne obuke relativno jednostavna sa aspekta korisnika programa. Potrebno je obezbedediti sledeće:

1. Govornu bazu u vidu audio fajlova (pored *wav* formata podržani su i *adc* i *raw*).
2. Label fajl sa granicama fonema. Videćemo kasnije da se procedura postavljanja granica može umnogome automatizovati.
3. Konfiguracioni fajl (zapravo samo treba modifikovati postojeći).
4. Ekstraktor obeležja (ili iskoristiti neki od ponuđenih).

Inicijalna obuka će napraviti tzv. *dictionary* fajl čije je *default* ime "schmm.dict". U njemu se nalaze svi podaci o napravljenim modelima, ali takođe i korišćeni ekstraktor i konfiguracioni fajl (njihov sadržaj) tako da kasnije, pri prepoznavanju, budu na raspolaganju. Kao što se moglo zaključiti iz algoritma, program će napraviti sve trifone kojih je bilo dovoljno u bazi, sve difone i od preostalih vektora sve monofone. Na ovaj način smo dobili skup modela od kojih možemo izgraditi bilo koji trelis, bez obzira na to koja se sekvenca fonema zahteva.

3.2 ML obuka

Posle inicijalne obuke u svim sistemima za prepoznavanje govora obično sledi tzv. *maximum likelihood* obuka. Primenjen je tzv. *expectation maximization* algoritam [8]. Sam algoritam je detaljno opisan u pomenutoj literaturi, jedino što je tamo dat algoritam za CDHMM model, odnosno za slučaj kada se smeše ne dele. Naš algoritam je, naravno, prilagođen primenjenom SCDHMM modelu.

4. PREPOZNAVANJE

Za potrebe prepoznavanja potrebno je obezbediti sledeće:

1. *dictionary* fajl koji sadrži HMM modele, dobijen inicijalnom i ML obukom.
2. Gramatiku za prepoznavanje, odnosno dijagram mogućih reči i dozvoljenih prelaza.
3. Fonetski transkriptor.
4. Postprocesor.

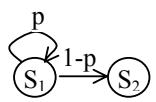
4.1 Akustički nivo

Kao što je već rečeno, koristi se model sa vezanim smešama. Što se tiče prelaza između stanja, zasad je podržan samo sekvencijalni model, dakle moguće je prelaziti samo redom iz stanja u susedno stanje. Ovde se misli na stanja u okviru jednog fonema. Kada se radi o prelazima između fonema i reči, moguće je izvesti proizvoljan graf prelaza, što ćemo videti posle kod zadavanja gramatike.

Koriste se standardne Gausove smeše sa dijagonalnom kovarijansnom matricom. Za računanje verovatnoća pojedinih stanja koriste se dva pristupa: računanje preko ponderisanih Gausovih smeša i metoda pobednik uzima sve [2]. Još jednom se ova druga metoda pokazala podjednako kvalitetnom u poređenju sa prvom, a obezbeđuje značajne uštede u vremenu.

4.2 Modelovanje trajanja stanja

Nakon izračunavanja verovatnoća pojedinih stanja u određenim trenucima, pronalaženje optimalne sekvence stanja vrši se Viterbijevim algoritmom. Poznat je način modelovanja trajanja pojedinih stanja preko verovatnoće ostanka u tom stanju i verovatnoće prelaska u drugo stanje, slika 4.1.

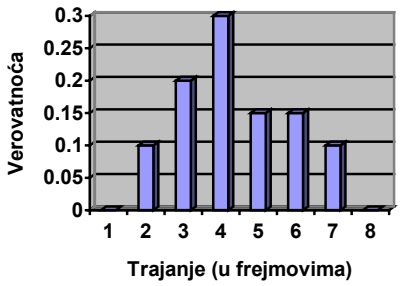


Slika 4.1: Klasično modelovanje trajanja stanja

Takođe je poznato da ovakav model daje eksponencijalnu raspodelu trajanja stanja što ne odgovara stvarnoj, prirodnoj raspodeli koja bi optimalno modelovala trajanje fonema.

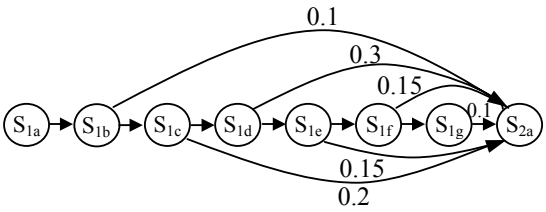
Zbog toga smo, pored ovog načina, podržali i model koji potpuno korektno modeluje trajanje stanja. Neka je

stanje S_1 imalo raspodelu trajanja u bazi za obuku, kao na slici 4.2.



Slika 4.2: Raspodela trajanja određenog stanja

Tada bi se ono moglo razbiti na nekoliko stanja, slika 4.3.



Slika 4.3: Prošireni model trajanja stanja

Neobeležene verovatnoće prelaza (one koje su unutar istog makro stanja) iznose 1. Vidimo da se ovakvim modelom postiže tačno onakva raspodela kakvu smo imali u bazi. Međutim takođe se može videti da se broj stanja drastično umnožava (za neka stanja i do nekoliko desetina puta). Jesu ovo ista stanja, po pitanju akustičkih obeležja, tako da ne zahtevaju zasebna računanja akustičke verovatnoće, ali ipak značajno opterećuju Viterbijev algoritam i usporavaju prepoznavanje. Korišćenjem ovakvog modela dobija se izvesno povećanje u tačnosti, ali odnos tačnosti i potrebnog procesorskog vremena nije baš povoljan. Stoga ga treba koristiti kada nam procesorska snaga nije problem.

Prikazani graf prelaza ograničava trajanje stanja S_1 strogo od 2-7 frejmova. U slučaju relativno male baze za obuku ovako stroga ograničenja mogu dovesti do pada tačnosti pri prepoznavanju, ako neko trajanje nije bilo zastupljeno u bazi. Zbog toga smo ostavili mogućnost da stanje traje manje od minimalnog i više od maksimalnog broja frejmova, ali se verovatnoće za tako nešto postavljaju na vrlo malu vrednost.

Iako je bolji od eksponencijalnog, ovaj metod još uvek ne tretira još jedan aspekt modelovanja trajanja stanja. Naime, ne modeluje se združena raspodela trajanja stanja, već se smatra da su raspodele verovatnoća trajanja pojedinih stanja statistički nezavisne. Zna se da je ovo pogrešno.

Na kraju se ukupna verovatnoća određenog stanja računa kao:

$$P = P_t\left(T = N \mid S_1\right)^K \prod_{n=1}^{N} P_a\left(S_1 \mid t = n\right) \qquad (4.1)$$

Gde je P_t verovatnoća da je stanje S_1 trajalo N frejmova, a P_a akustička verovatnoća stanja S_1 u trenutku n. Ukoliko bi K bilo jednako jedan, ovo bi bio klasičan slučaj, sa pokrićem u teoriji verovatnoće. Međutim, dajući ovom koeficijentu vrednosti veće od jedan, mi povećavamo težinu verovatnoće trajanja u odnosu na akustičku, i eksperimenti su pokazali da ovo rezultuje u povećanju tačnosti prepoznavanja. Ovaj koeficijent se može zadati u konfiguracionom fajlu.

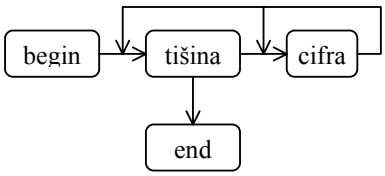
4.3 Gramatika

Naš sistem može da se koristi za prepoznavanje reči iz malih i srednjih rečnika. U ovim slučajevima uobičajeno je korišćenje regularne gramatike. Gramatika se može zadavati na dva načina:

1. Preko dijagrama prelaza.
2. Korišćenjem EBNF forme (*Extended Backus Naur Form*) [4].

4.3.1 Zadavanje gramatike preko dijagrama prelaza

Recimo da hoćemo da definišemo jednostavnu gramatiku kojom će se prepoznati proizvoljan broj cifara. Grafički prikaz odgovarajućeg dijagrama prelaza bi izgledao kao na slici 4.4.



Slika 4.4: Dijagram prelaza gramatike za cifre

Dakle dozvoljeni su kako direktni prelazi iz cifre u cifru, tako i prelazi preko pauze. Veoma je važno ovo definisati da bi se pravili optimalni modeli. Ukoliko bismo predvideli samo prelaze preko tišine, dobili bismo kraći i brži trelis, ali bi tačnost sistema značajno opala ako spiker ne bi pravio eksplicitne pauze između reči.

Ovaj dijagram se zadaje u testualnom fajlu i imao bi ovakav oblik:

```
main
{
   begin -> tisina;
   tisina -> cifra;
   cifra -> cifra;
   cifra -> tisina;
   tisina -> end;
}
cifra
{
   begin -> nula;
   begin -> jedan;
   nula -> end;
   jedan -> end;
}
tisina { "_" }
nula { "NULA" }
jedan { "JEDAN" }
```

Identifikatori "cifra", "tisina", "nula" i "jedan" su takozvana pravila i odlikuju se ili novim dijagramom prelaza (kao "cifra") ili sekvencom koju predstavljaju. Rezervisani identifikator "main" opisuje glavno pravilo u gramatici. Rezervisani identifikatori "begin" i "end" definišu početak i kraj svakog dijagrama prelaza.

4.3.2 Zadavanje gramatike korišćenjem EBNF forme

Druga metoda je daleko više korišćena i fajl koji je definiše je daleko manji. Racimo da hoćemo da definišemo gramatiku kojom korisnik želi da bira lokal. Želimo da mu dozvolimo neke uvodne sekvence, npr. "Molim Vas", "Molio bih" i "Treba mi". Ovo radimo na sledeći način:

```
uvod = (MOLIM VAS)|(MOLIO BIH)|(TREBA
MI);
```

Zatim korisnik može da kaže reč "Lokal" i potom niz od nekoliko cifara. Definicija cifre (0-3):

```
cifra = NULA | JEDAN | DVA | TRI;
```

Konačna gramatika glasi:

```
main                                    =
[gar][$uvod][LOKAL]<$cifra>[gar];
```

Uglaste zagrade znače da je to opciona sekvenca, a zagrade "<>" znače da se ta sekvenca ponavlja jedan ili više puta. "gar" je skraćeno od *garbage*, ali se ne radi ni o kakvoj rezervisanoj reči. Ovo bi trebao da bude model koji je obučavan na šumove koji su se nalazili u bazi i služi da "proguta" takve pojave u prepoznavanom govoru. Pored već viđenih znakova: "|", "[]" i "<>", mogu se koristiti i "{}" zagrade koje znače da se sekvenca ponavlja nula ili više puta. Još jednom vidimo da je konvencija kompatibilna sa HTK, jedino što nisu podržani simboli "<<" i ">>".

Parser EBNF forme interno prevodi ovaj kompaktniji zapis u dijagram prelaza stanja, jer mu je lakše da od te tačke krene dalje u pravljenje trelisa.

Treba zapaziti da u gramatici nije eksplicitno zadavano postojanje tišine, kao ni da li prelazi između reči treba da budu direktni ili preko tišine (odnosno da li se zahteva da korisnik pravi pauze između reči). Zadavanjem odgovarajućeg parametra u konfiguracionom fajlu, prepoznavaču se može reći da ovo odradi automatski. Dakle prelaz između bilo koje dve reči može biti direktan ili preko tišine. Ova opcija umnogome rasterećuje korisnika.

4.4 Fonetski transkriptor

Kao što se moglo primetiti, prilikom zadavanja gramatike koristi se ortografski zapis reči koje treba prepoznavati. Doduše u srpskom jeziku ortografski i fonetski zapis su skoro identični (mada, videćemo, ima i tu odstupanja), ali je za većinu jezika ovo neophodan korak. Fonetska transkripcija reči se odvija u dva koraka:

1. Traženje ortografske reči u rečniku izgovora.
2. Primena skupa pravila na ortografski tekst.

4.4.1 Traženje reči u rečniku

Ukoliko se želi koristiti ova opcija mora se obezbediti fajl u kome će se naći parovi ortografskih i fonetskih zapisa pojedinih reči. Na primer:

```
NULA    N U L A
JEDAN   J E Do De A N
JEDAN   J E A N
```

Svaki red opisuje novu reč. Treba zapaziti da se u fonetskom delu fonemi odvajaju razmakom. Vidimo da je podržan višestruk izgovor pojedinih reči sa istim ortografskim zapisom. Ovi slučajevi se, kasnije, na nivou gramatike razvrstavaju na nezavisne grane za prepoznavanje. Pri detranskripciji ponovo se vraćamo na istovetan ortografski zapis, bez obzira kroz koju verziju izgovora je prepoznavač prošao.

Može se videti i da smo pri realizovanju našeg prepoznavača okluziju i eksploziju (Do i De) tretirali kao odvojene foneme, što je moguć, ali ne i obavezan pristup.

4.4.2 Primena skupa pravila

Ukoliko se radi o jeziku sa strogim pravilima konverzije ortografskog u fonetski zapis, prethodni pristup je preobiman i zahteva nepotrebno pravljenje velikog fajla sa izgovorima. U takvim slučajevima pogodnije je primeniti određeni skup pravila na ortografski tekst i na taj način dobiti fonetski zapis. Evo kako bi izgledao takav fajl za srpski jezik (skraćena verzija):

```
vowels { A E I O U }
rules
{
   // majica -> maica
   vowel J I -> vowel I;
   // petnaest -> petnajst
   N A E S T -> N A J S T;
   // razdvajanje ploziva i afrikata na
   // okluziju i eksploziju
   P -> Po Pe;
   C -> Co Ce;
   ...
}
```

Na reč u ortografskom zapisu se primenjuju sva navedena pravila i to navedenim redom (dosta bitno u nekim slučajevima). Ovakvim skupom pravila bi se mogao u dobroj meri opisati i jezik sa velikim brojem izuzetaka u izgovoru, a samo izuzetke obraditi primenom rečnika. Takva logika je i primenjena: ako reč nije nađena u rečniku na nju se primenjuju navedena pravila.

Mana pristupa sa pravilima je što se ne može realizovati višestruki izgovor jedne reči. Ako tako nešto želimo moramo pribeći rečniku.

4.5 Detranskripcija

Detranskripcija je takođe zanimljiv proces koji sledi nakon prepoznavanja. Neka, na primer, iz prepoznavača dobijemo fonetsku sekvencu:

```
N U L A J E A N Do De V A
```

Korisnika će naravno interesovati reči u originalnom, ortografskom, zapisu, pogodno razdvojene. Odnosno:

```
NULA JEDAN DVA
```

Da bismo dobili ovu sekvencu potrebno je na fonetsku sekvencu primeniti određeni skup pravila, koji bi u konkretnom slučaju mogao da glasi:

```
rules
{
   J E Do De A N -> "JEDAN ";
   Do De V A -> "DVA ";
   J E A N -> "JEDAN ";
   N U L A -> "NULA ";
}
```

Ovde postoji mali problem ukoliko je neka manja reč podskup veće (npr. da je u konkretnom slučaju postojala i reč "JE"). Tada bi moglo da se desi da se manje pravilo prvo primeni, a da deo veće reči ostane u fonetskom obliku, što dovodi do greške. Zbog toga su gornja pravila sortirana po veličini fonetskog dela, tako da je ova pojava izbegnuta.

Ovaj skup pravila za detranskripciju se generiše automatski, prilikom transkripcije zadate gramatike, tako da korisnik ne mora da razmišlja o tome.

4.6 Postprocesor

Ako bismo ostali samo na detranskripciji, mogli bismo na kraju da dobijemo neki rezultat tipa:

```
_ NULA _ gar _ JEDAN _ gar _
```

Korisnika najčešće ne interesuju informacije o tišini i pojavi šumova u prepoznatoj sekvenci, pa bi morao ručno da čisti dobijenu sekvencu od ovih viškova. Zbog toga je u prepoznavaču predviđena opcija koja ovo uradi pre vraćanja rezultata korisničkoj aplikaciji. Dovoljno je definisati nekoliko jednostavnih pravila koja će to da odrade, a u konkretnom slučaju ona bi glasila:

```
rules
{
   _ ->;
   gar ->;
}
```

5. AUTOMATSKO LABELIRANJE

Kao što je rečeno, deo programa za obuku zahteva prisustvo labeliranih zvučnih sekvenci, odnosno label fajl sa korektno postavljenim granicama između glasovnih jedinica. Jasno je da bi ovaj zadatak bio veoma obiman ukoliko bi morao da se u potpunosti obavi ručno. Zbog toga je obezbeđena opcija koja automatski postavlja granice između fonema, ako su obezbeđeni inicijalni akustički modeli (*dictionary* fajl) i label fajl sa korektnom fonetskom transkripcijom. Ovo se radi tako što se izgradi trelis koji će forsirati prepoznavač da prođe kroz sve foneme u labeli, a zatim se pusti Viterbijev algoritam da nađe optimalni put. Pronađena segmantacija se upisuje u label fajl.

Naravno, nije teško primetiti "mali" paradoks: odakle nam *dictionary* fajl? Pa, neko inicijalno postavljanje granica na relativno malom podskupu baze će se morati odraditi ručno. Uz pomoć relativno malog broja ručno odrađenih labela može se izgraditi neki inicijalni model. On će biti relativno loš, ali može poslužiti da se u sledećem koraku granice automatski postave. Dalje se može postupiti na nekoliko načina.

Jedan od načina je da se pusti automatsko labeliranje cele baze, a zatim da se opet izvrši obuka na tako labeliranoj celoj bazi, uz odbacivanje *outlier*-a prilikom obuke. U sledećoj iteraciji se isto radi, ali sa novim modelom.

Drugi način je da se nakon prvog kruga automatske labelacije uoče problemi, odnosno mesta gde labelator greši. Zatim se obradi novi podskup baze u kojem će biti zastupljene one sekvence sa problematičnim kombinacijama fonema. Ponovo se obuči model na proširenom podskupu itd. Može se ići i na neku kombinaciju ova dva načina.

Prilikom automatskog labeliranja treba obratiti pažnju na nekoliko detalja:

- Modeli treba da imaju malo smeša (ne više od dve).
- *frame_duration* parametar je dobro smanjiti, kako bi se postigla preciznija segmentacija.
- Preporučljivo je podeliti govornike u nekoliko različitih grupa, barem na muškarce i žene.
- Za modelovanje trajanja fonema bolje je koristiti drugi pristup, preko prave raspodele trajanja.

6. FAZA ISPITIVANJA

Sve navedene metode obuke su u potpunosti debagirane, istestirane i operativne. Pored njih postoje još neki načini obuke i metode prepoznavanja koje do trenutka pisanja rada nisu do kraja implementirane ili dovoljno ispitane. Ipak ćemo navesti nekoliko:

6.1 Korektivni trening

Korektivni ili diskriminacioni trening bi trebao da dodatno poboljša modele dobijene ML obukom, u smislu da se oni još više razdvoje i da prave još bolju distinkciju između različitih modela. Primenjeni su algoritam i logika veoma slični onim u [2], s tim što se sad međusobno porede modeli fonema u kontekstu, a ne kompletne reči. Ovo dovodi do niza problema u ovakvom pristupu:

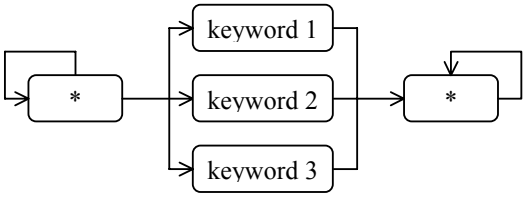
1. Ne možemo praviti distinkciju između dva modela, ako im se u osnovi nalaze isti ili istorodni subfonemi (npr. A0 i A1).
2. Čak i kad osnovni fonemi nisu istorodni može neki model veoma ličiti na drugi. Na primer model **A2**+M (desni subfonem od **A** čiji je desni kontekst **M**) i model A-**M0** (levi subfonem od **M** čiji je levi kontekst **A**) veoma liče i opisuju delove signala koji se često preklapaju.
3. Okluzije svih bezvučnih ploziva i afrikata imaju praktično ista obeležja. Isto važi za okluzije zvučnih ploziva i afrikata.
4. U zavisnosti od kanala i govornika mogu i mnogi drugi fonemi ličiti jedni na druge. Na primer u telefonskom kanalu, zvučno - bezvučni parovi suglasnika veoma liče zbog odsecanja donjih 300Hz. Ukoliko je govornik govorio na neki atipičan i/ili ležeran način može se desiti da liče fonemi za koje nikada ne bi rekli da mogu ličiti. Recimo "D" u ležerno izgovorenoj reči "JEDAN", van svih očekivanja, može veoma da liči na "R".

5. Kad sve navedeno uzmemo u obzir dolazimo do zaključka da ima smisla praviti dodatnu distinkciju samo između modela koji se već dobro razlikuju, što obesmišljava celu ideju.

Iz navedenih razloga će biti potrebno preispitati pristup u kojem bi se pravila dodatna distinkcija na nivou fonema. Možda će se ići na korektivnu obuku koja će dodatno optimizovati sistem za pojedine gramatike, dakle praviti distinkcije na nivou reči, što ima više smisla.

6.2 Word spotting ili upotreba džokera

Pod "džokerom" se podrazumeva neki model koji bi mogao u dovoljno dobroj meri da opiše bilo koji fonem, ali da verovatnoća prolaska kroz njega bude manja od verovatnoće prolaska kroz korektan fonem. Ukoliko bismo imali na raspolaganju takav model mogli bi da napravimo sledeću gramatiku:



Slika 6.1: Dijagram prelaza *word-spotter*-a

U slučaju da je u test sekvenci izgovorena neka od ključnih reči, optimalna staza kroz ovakvu gramatiku bi vodila preko te ključne reči i ona bi bila prepoznata ("uočena"). U suprotnom, najveću verovatnoću će imati ostanak u "džoker" stanju.

U programu je implementirana opcija koja pravi ovakve modele, ali je taj deo još uvek u fazi testiranja.

7. ZAKLJUČAK

U radu je opisan jedan kompletan programski paket za prepoznavanje kontinualnog govora, koji se bazira na prepoznavanju fonema u kontekstu.

Primenjene su mnoge standardne metode za obuku i prepoznavanje, a isprobane su i neke nove. Poboljšani način modelovanja trajanja stanja može značajno da poveća tačnost prepoznavanja ukoliko nam procesorska snaga nije problem u konkretnoj primeni. Primena skupa pravila za fonetsku transkripciju oslobađa korisnika potrebe za velikim rečnikom sa izgovorima, što ima primenu u mnogim jezicima. Algoritam po kojem se sprovodi inicijalna obuka u potpunosti rasterećuje korisnika brige oko toga koji će fonemi biti napravljeni i garantuje mogućnost izgradnje proizvoljne gramatike. *Expectation maximization* metoda za SCDHMM model, primenjena u ML obuci, razvijena je na osnovu iste metode predložene u literaturi za CDHMM model. Korektivni trening, mada još uvek u fazi ispitivanja, obećava poboljšanje performansi i dodatnu funkcionalnost prepoznavača.

Format konfiguracionog fajla omogućava zadavanje proizvoljno složenih struktura i koncentrisanje svih parametara potrebnih za obuku i prepoznavanje na jednom mestu.

Automatsko keširanje obeležja na disk rasterećuje korisnika programa ručnog obavljanja ovog posla.

U mnogim delovima ovaj paket je kompatibilan ili bar liči na poznati HTK. Rezultati koji se dobijaju upotrebom ovog programskog paketa ne samo da su uporedivi sa onim dobijenim upotrebom HTK, već ih i nadmašuju.

Ceo program napisan je u C++ programskom jeziku, u razumljivom i elegantnom obliku, ali istovremeno i maksimalno optimizovanom, tamo gde je to bitno, tako da je spreman i za komercijalnu upotrebu.

LITERATURA

[1] D. Pekar, R. Obradović, *C++ Library for Digital Signal Processing - slib*, Telfor, Beograd, Novembar 2001.

[2] D. Pekar, R. Obradović, V. Delić, *Connected Words Recognition*, Telfor, Beograd, Novembar 2001.

[3] R. Obradović, D. Pekar, S. Krčo, V. Delić, V. Šenk, *A Robust Speaker Independent CPU Based Speech Recognition System*, *Eurospeech*, Vol.6. pp. 2881-2884, Budapest, September 1999.

[4] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, P. Woodland, *The HTK Book*, 1995-1999, Microsoft Corporation

[5] Rabiner, B.H. Juang, *Fundamentals of Speech Recognition,* Prentice-Hall, New Jersey, 1993.

[6] Mikko Kurimo, *Application of Learning Vector Quantization and Self-Organizing Maps for training continuous density and semi-continuous Markov models*, Licentiate's Thesis, Helsinki University of Technology, 1994.

[7] J.C. Junqua, J.P. Haton, *Robustness In Automatic Speech Recognition,* Kluwer, 1993.

[8] Perry Moerland, *Mixture Models for Unsupervised and Supervised Learning*, Ph.D. thesis, Swiss Federal Institute of Technology, 2000.

[9] S. Krčo, *Prilog metodama za prepoznavanje izolovano izgovorenih reči,* Magistarski rad, ETF Beograd, 1997.

[10] N. Draper, H. Smith, *Applied Regression Analysis,* Wiley, 1981.

[11] T. Back, H. P. Schwefel, *An Overview of Evolutionary Algorithms for Parameter Optimization,* University of Dortmund, 1993.

[12] P. Yao, R. Leinecker, *Visual C++ 6 Bible,* IDG Books Worldwide, Foster City, USA, 1999.

[13] A. Stepanov, *Standard Template Library,* Hewlet Packard, 1995.

ABSTRACT

This paper shortly presents a program package for continuous speech recognition, named AlfaNumCASR. This HMM-based phoneme in context recognizer is successful with small and medium dictionaries. AlfaNumCASR is a product of several years of R&D in ASR topic, starting from isolated words recognition, over connected words, to continuous speech recognition. AlfaNumCASR employs some unique procedures described in this paper. The whole program is written in C++ programming language, and is fully developed by the authors. Software is in its largest part independent of the platform or the operating system. It includes two libraries developed in last two years by the same authors. Those are **slib** library for digital signal processing and general-purpose **an_misc** library.