KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMATIKA

DARIUS SABALIAUSKAS

LIETUVIŲ KALBOS ATPAŽINIMAS iOS ĮRENGINIUOSE

Magistro darbas

Vadovas
doc., R. Maskeliūnas
Parašas:_____
Data: _____

KAUNAS, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMATIKA

DARIUS SABALIAUSKAS

LIETUVIŲ KALBOS ATPAŽINIMAS iOS ĮRENGINIUOSE

Magistro darbas

Vadovas
doc., R. Maskeliūnas
Parašas:_____
Data: _____

Recenzentas
doc., V. Rudžionis
Parašas:_____
Data: _____

Studentas
D. Sabaliauskas
Parašas:_____
Data: _____

KAUNAS, 2014

AUTORIŲ GARANTINIS RAŠTAS
DĖL PATEIKIAMO KŪRINIO

20.. - ..... - ..... d.
Kaunas

Autoriai,_____
(vardas, pavardė)

_____,

patvirtina, kad Kauno technologijos universitetui pateiktas baigiamasis bakalauro (magistro) darbas (toliau vadinama – Kūrinys)_____
(kūrinio pavadinimas)
_____

pagal Lietuvos Respublikos autorių ir gretutinių teisių įstatymą yra originalus ir užtikrina, kad

1) jį sukūrė ir parašė Kūrinyje įvardyti autoriai;
2) Kūrinys nėra ir nebus įteiktas kitoms institucijoms (universitetams) (tiek lietuvių, tiek užsienio kalba);
3) Kūrinyje nėra teiginių, neatitinkančių tikrovės, ar medžiagos, kuri galėtų pažeisti kito fizinio ar juridinio asmens intelektinės nuosavybės teises, leidėjų bei finansuotojų reikalavimus ir sąlygas;
4) visi Kūrinyje naudojami šaltiniai yra cituojami (su nuoroda į pirminį šaltinį ir autorių);
5) neprieštarauja dėl Kūrinio platinimo visomis oficialiomis sklaidos priemonėmis.
6) atlygins Kauno technologijos universitetui ir tretiesiems asmenims žalą ir nuostolius, atsiradusius dėl pažeidimų, susijusių su aukščiau išvardintų Autorių garantijų nesilaikymu;
7) Autoriai už šiame rašte pateiktos informacijos teisingumą atsako Lietuvos Respublikos įstatymų nustatyta tvarka.

Autoriai

_____		_____	
(vardas, pavardė)	(parašas)		
_____		_____	
(vardas, pavardė)	(parašas)		
_____		_____	
(vardas, pavardė)	(parašas)		
_____		_____	
(vardas, pavardė)	(parašas)		

SANTRAUKA

Sabaliauskas D. Lietuvių kalbos atpažinimas iOS įrenginiuose: Informatikos magistro baigiamasis darbas / vadovas doc., R. Maskeliūnas; Kauno technologijos universitetas, Informatikos fakultetas, Multimedijos inžinerijos katedra. Kaunas, 2014. 52 p.

Šiuolaikiniame pasaulyje vis daugiau žmonių naudoja išmaniuosius telefonus, kurie perima vis daugiau su kompiuterio atliekamo darbo (el. pašto tikrinimas, apsipirkimas internetu ir t.t.). Šiuose įrenginiuose vis daugiau funkcijų galima atlikti balsu (atidaryti programėles ir kt.), tačiau kol kas tik anglų ir keletu kitų kalbų. Todėl šiame darbe bus nagrinėjamas lietuvių kalbos atpažinimo uždavinys iOS platformai (viena iš pagrindinių išmaniųjų telefonų ir planšetinių kompiuterių platformų), kuri yra naudojama mobiliuose Apple įrenginiuose.

Šiame darbe nagrinėjamas CMU Sphinx ir Julius bibliotekų panaudojimas iOS įrenginiuose atpažįstant lietuvių kalbą. Tyrimui buvo sukurtas LSR karkasas paslepiantis CMU Sphinx ir Julius bibliotekų realizacijos ypatumus po Objective-C kalbos sąsaja ir testinė programėlė naudojanti šį karkasą eksperimentų atlikimui. Be to akustinių modelių, žodyno ir kalbos modelio kūrimui buvo sukurtas Perl skriptų rinkinys, kuris automatizavo kūrimo darbą.

Tyrimui buvo naudojamas skaičių nuo 0 iki 9 garsynas (100 diktorių iš jų 50 vyrų ir 50 moterų, 10349 įrašai, kurių bendra trukmė 2 valandos 45 minutės 28 sekundės) ir analizuota, koks atpažinimo tikslumas ir greitaveika yra su tokiu nedideliu 10 žodžių žodynu atpažįstant pavienius skaičius. Eksperimentų metu paaiškėjo, kad Julius biblioteka su standartiniais nustatymais ir pagal standartinį aprašymą sukurtais akustiniais modeliais (naudojant HTK) ir tiksliau ir greičiau atpažino testinius įrašus. Julius bibliotekos mažiausias pasiektas WER buvo 3.5%, o CMU Sphinx 12.4%. Taip pat Julius bibliotekos geriausias vidutinis 1 sekundės įrašo atpažinimo laikas 50x16 akustiniam modeliui buvo 0.026 s, kai analogiško akustinio modelio CMU Sphinx bibliotekai vidutinis 1 sekundės įrašo atpažinimo laikas buvo 0.055 s. Šie rezultatai rodo, kad Julius bibliotekos standartinis akustinių modelių kūrimo ir atpažinimo procesas yra ženkliai geresnis nei CMU Sphinx ir Julius biblioteką galima naudoti realiose sistemose kone iškart, kai tuo tarpu norint naudoti CMU Sphinx reikia papildomų tiek akustinių modelių kūrimo tiek atpažinimo proceso optimizavimo žinių.

Tačiau nepaisant to, kad CMU Sphinx eksperimentiniais rezultatais nusileido Julius, ilgesnėje perspektyvoje atrodo patrauklesnis lietuvių kalbos atpažinimui iOS įrenginiuose, nes palaiko UTF-8 koduotę ir turi integruotą akustinių modelių apmokymo mechanizmą, kai tuo tarpu Julius nepalaiko UTF-8 koduotės ir akustinius modelius reikia kurti su HTK įrankiu. Todėl norint perkelti ir akustinių modelių kūrimą ar adaptavimą į iOS platformą su CMU Sphinx būtų paprasčiau, nei su Julius (HTK nepalaiko iOS).

Taip pat atlikus eksperimentus paaiškėjo, kad nedidelio žodyno pavienių žodžių atpažinimo uždaviniai gali būti nesunkiai sprendžiami iOS įrenginiuose realiu laiku, nes atpažinimo greitaveika yra pakankama. Greičiausiame iOS įrenginyje iPad Air vidutinis 1 sekundės įrašo atpažinimas truko vos 0.026 s Julius bibliotekai ir 0.055 s CMU Sphinx bibliotekai.

SUMMARY

Sabaliauskas D. Lithuanian speech recognition in iOS devices: Informatics Master Thesis/ Supervisor doc., R. Maskeliūnas: Kaunas University of Technology, Faculty of Informatics; Cathedral of Multimedia Engineering. Kaunas, 2014. 52 p.

Nowadays more and more people use smartphones which replaces more word done with personal computer (e-mail checking, e-shopping, etc.). In these devices more and more functions could be done with voice (open apps and other), but still only in english and some other languages. Therefore, in out work we will investigate Lithuanian speech recognition task in iOS (one of the major smartphones and tablets platforms), which runs in Apple's mobile devices.

In this work we investigate CMU Sphinx and Julius libraries use in iOS devices for Lithuanian speech recognition. For this task LSR framework was created which encapsulated CMU Sphinx and Julius realization nuances under Objective-C interfaces and test app which was using this framework to perform experiments. Further more, for acoustic models, dictionary and language model creation automatization collection of Perl scripts was created.

Experiments were performed with numbers from 0 to 9 corpus and recognition accuracy and speed were investigated. After experiments it was clear that Julius library outperformed CMU Sphinx. WER for Julius was 3.5% and WER for CMU Sphinx was 12.4%. Average recognition time for 1 second of record for 50x16 acoustic models was 0.026 s for Julius (acoustic models created with HTK) and 0.055 s for CMU Sphinx. Despite that Julius outperformed CMU Sphinx acoustic models wasn't exact, but created with standard descriptions and recognition was run with default libraries configurations. That shows that Julius out of the box is better that CMU Sphinx out of the box, and Julius can be used for real systems that way, bu CMU Sphinx can't. CMU Sphinx needs more knowledge in speech recognition because you need optimize acoustic models creation and recognition configuration.

Although, CMU Sphinx was outperformed by Julius is better suited for Lithuanian speech recognition in long term, because it supports UTF-8 and have integrated acoustic models creation mechanism. Julius in this chase do not support UTF-8 and acoustic models are created with HTK. Therefore, in order to move acoustic models creation or adaptation to iOS with CMU Sphinx is easier than with Julius (HTK do not support iOS).

After experiments become clear that small vocabulary separated word recognition problems could be solved in iOS devices in real time, because recognition time for this kind of task is sufficient. On fastest tested device iPad Air average recognition time for 1 second of record was only 0.026 s for Julius library and 0.055 s for CMU Sphinx library.

TURINYS

TERMINŲ IR SANTRUMPŲ ŽODYNAS....8
LENTELIŲ SĄRAŠAS....9
PAVEIKSLŲ SĄRAŠAS....10
1.ĮVADAS....11
1.1.Tyrimo sritis, objektas ir problematika....11
1.2.Tyrimo tikslas ir uždaviniai....11
1.3.Dokumento struktūra....11
2.ANALITINĖ DALIS....12
2.1.Panaudojimo sritys....12
2.2.Esamų sprendimų analizė....13
2.3.Signalo paruošimas....13
2.3.1.Akustinio aido slopinimas....13
2.3.2.Triukšmų slopinimas....14
2.4.Melų skalės kepstro koeficientai....14
2.5.Akustiniai modeliai....15
2.6.Kalbos modelis....16
2.7.Statistiniai metodai....16
2.7.1.Paslėptas Markovo modelis....16
2.7.2.Viterbi algoritmas....20
2.8.Neuroniniai tinklai....21
2.8.1.Naudojimas....21
2.8.2.Pagrindinis principas....21
2.8.3.Tinklų topologijos....23
2.8.4.Apmokymas....24
2.9.Dinaminis laiko skalės kraipymas....26
2.10.Apibendrinimas ir išvados....29
3.PROGRAMINĖS ĮRANGOS PROJEKTAVIMAS....30
3.1.Planuojamas sprendimas....30
3.2.Naudojamų kalbos atpažinimo įrankių apžvalga....30
3.2.1.CMU Sphinx....30
3.2.2.Julius....30
3.2.3.HTK Toolkit....30
3.3.Versijų kontrolė....30
3.4.Sistemos architektūra....31
3.5.Sprendimo realizacija....33
3.5.1.Reikalingų failų generavimas....33
3.5.2.Atpažinimo karkasas....33
3.5.3.Kalbos atpažinimas....34
3.6.Apibendrinimas ir išvados....35
4.EKSPERIMENTINĖ DALIS....36
4.1.Tyrimo eigos aprašymas....36
4.1.1.Garsynas....36
4.1.2.Automatinis kalbos atpažinimo tikslumo vertinimas....36
4.1.3.Įrenginiai....36
4.1.4.Eksperimento eiga....37
4.2.Apmokymo duomenų dydžio ir akustinio modelio įtaka atpažinimo tikslumui....37
4.3.Atskirų skaičių atpažinimo tikslumas....40

4.4.Įrenginio įtaka atpažinimo greičiui....41
4.5.Eksperimentų išvados....44
5.DARBO REZULTATAI IR IŠVADOS....45
LITERATŪRA....45
PRIEDAI....48
1.CMU SPHINX KONFIGŪRACINIS FAILAS....48
2.HTK KONFIGŪRACINIS FAILAS....51
3.CSV FAILO PAVYZDYS....51
4.JULIUS FAILŲ KOREKCIJOS....51

TERMINŲ IR SANTRUMPŲ ŽODYNAS

PMM – paslėptas Markovo modelis (angl. *Hidden Markov Model*).
UTF-8 – simbolių kodavimo sistema unikodo simboliams (tarp jų ir lietuviškiems).
ASCII – simbolių kodavimo sistema grįsta angliškų žodynu.
LSR – lietuvių kalbos atpažinimas (angl. Lithuanian speech recognition).
HTK – paslėptų Markovo modelių rinkinys (angl. *Hidden Markov Model Toolkit*).
MSKK – melų skalės kepstrų koeficientai (angl. *MFCC – Mel frequency cepstral coefficients*).
ANN – neuroninis tinklas (angl. *Artificial Neural Network*).
FNN – neuroninis tinklas be grįžtamųjų ryšių (angl. *Feed-forward neural network*).
RNN – neuroninis tinklas su grįžtamaisiais ryšiais (angl. Recurent *neural network*).
DLSK – dinaminis laiko skalės kraipymas (angl. *Dynamic time warping*).
WER – žodžio klaidos procentas – WER (angl., *Word Error Rate*).

LENTELIŲ SĄRAŠAS

2.2.1 lentelė Esamų sistemų palyginimas.....13
4.1.3.1 lentelė. Įrenginių sąrašas.....36
4.2.1 lentelė. WER skirtingiems akustiniams modeliams ir duomenų imtims.....38
4.3.1 lentelė. Julius skaičių atpažinimo klaida.....40
4.3.2 lentelė. CMU Sphinx skaičių atpažinimo klaida.....41
4.4.1 lentelė. Julius bibliotekos vidutinis atpažinimo laikas 1 sekundei įrašo.....42
4.4.2 lentelė. CMU Sphinx bibliotekos vidutinis atpažinimo laikas 1 sekundei įrašo.....43

PAVEIKSLŲ SĄRAŠAS

2.5.1 pav. Fonemų akustiniai modeliai....16
2.7.1.1 pav. Paslėptas Markovo modelis....17
2.8.2.1 pav. Biologinis neurono modelis....22
2.8.2.2 pav. Matematinis neurono modelis....22
2.8.2.3 pav. Aktyvacijos funkcijų pavyzdžiai....23
2.8.3.1 pav. Neuroninio tinklo be grįžtamųjų ryšių (FNN) ir su grįžtamaisiais ryšiais (RNN) topologijos....23
2.8.4.1 pav. Neuroninio tinklo klaidos skleidimas atgal....24
2.9.1 pav. Dviejų nuo laiko priklausomų sekų sutapatinimas. Sutapatinti taškai pažymėti rodyklėmis 26
2.9.2 pav. (a) Skirtingumo įverčių matrica C (b) sukauptų skirtingumo įverčių matrica D su optimaliu sutapatinimo keliu p* (balta linija)....27
3.3.1 pav. Repozitorijų schema....31
3.4.1 pav. Sistemos architektūra....31
3.4.2 pav. Analizės modulis....31
3.4.3 pav. Atpažinimo bibliotekos modulis....32
3.4.4 pav. UML klasių diagrama LSR karkasui....32
3.5.2.1 pav. Nuoseklus atpažinimas....34
3.5.2.2 pav. Lygiagretus atpažinimas....34
3.5.3.1 pav. Atpažinimo schema....35
4.2.1 pav. Atpažinimo bibliotekų WER pagal akustinius modelius....39
4.2.2 pav. Atpažinimo bibliotekų WER pagal duomenų imtį....39
4.3.1 pav. Julius skaičių atpažinimo klaida....40
4.3.2 pav. CMU Sphinx skaičių atpažinimo klaida....41
4.4.1 pav. Julius bibliotekos atpažinimo greitis....42
4.4.2 pav. CMU Sphinx bibliotekos atpažinimo greitis....43
4.4.3 pav. CMU Sphinx ir Julius bibliotekų atpažinimo greičių palyginimas 50x16 akustiniam modeliui....44

1. ĮVADAS

Tobulėjant technologijoms tobulėja ir žmogaus-kompiuterio sąsajos. Kalba yra viena natūraliausių ir intuityviausių sąsajų, kurią žmogus linkęs naudoti, tačiau, kad ir kaip būtų nesudėtinga žmogui suprasti žmogų, kompiuteriui šis supratimo uždavinys yra labai sudėtingas. Taip yra dėl pakankamai daug priežasčių, štai keletas jų: natūrali žmogaus kalba yra analoginis signalas, kuriame balsiai, priebalsiai ir iš jų sudaryti žodžiai sudaro nenutrūkstamą signalą iš kurio pakankamai sunku nustatyti žodžio pradžią ir pabaigą; ta pati frazė ar žodis ištarti skirtingų žmonių dėl akcentų, tarmių ir balso tembro gali turėti visiškai skirtingą signalo formą; natūrali aplinka pilna pašalinių triukšmų; tą patį reiškiantį sakinį galima labai įvairiai pasakyti, tiek pridedant neesminių žodžių (pvz. malonybės), tiek nekalbinių intarpų (pvz. mikčiojimas, kostelėjimas), tokiu atveju nors loginė sakinio prasmė nepakinta signalas yra visiškai skirtingos struktūros. Taigi kaip matome kalbos atpažinimo uždavinys yra labai sudėtingas ir apimantis pačio kalbos signalo pirminį apdorojimą (triukšmų šalinimą), parametrų išskyrimą iš signalo ir lingvistinių vienetų (žodis, skiemuo, fonema) išskyrimą, bei pačia loginę analizę, ką norėta pasakyti.

Darbe bus apsiribota atskirų žodžių, kaip komandų atpažinimu, tačiau nors ir supaprastėja kalbos atpažinimo uždavinys iki atskirų žodžių atpažinimo, tai kad kalbos atpažinimo modulis turės veikti iOS išmaniuosiuose įrenginiuose su ribotais skaičiavimo resursais ir be interneto prieigos, kelia naujus iššūkius. Sukurtu kalbos atpažinimo moduliu bus tiriamas nagrinėjamų balso komandų atpažinimo tikslumą bei praktinio panaudojimo galimybės.

1.1. Tyrimo sritis, objektas ir problematika

Automatinis kalbos atpažinimas yra pakankamai senas uždavinys, pasaulyje pirmieji bandymai spręsti šį uždavinį buvo 1950 m., o Lietuvoje tik 1970 m. (taigi Lietuvoje yra atsiliekama apie 20 m. nuo pasaulinės praktikos).

Automatinio kalbos atpažinimo uždavinys yra sudėtingas ir vis dar nėra sukurta efektyvaus ir nepriklausomo nuo kalbėtojo metodo nenutrūkstamai kalbai atpažinti.

Kalbos atpažinimas yra audio signalo analizė išgaunant šį signalą aprašančius parametrus, kuriuos sistema interpretuoja ir pateikia spėjimą, koks tai galėtų būti žodis ar žodžių seka.

1.2. Tyrimo tikslas ir uždaviniai

Tikslas – išanalizuoti, kaip lietuvių kalbos atpažinimas veikia iOS įrenginiuose panaudojant Julius ir CMU Sphinx bibliotekas.

Uždaviniai:

1. Ištirti, kuri balso atpažinimo biblioteka (Julius ar CMU Sphinx) yra labiausiai tinkama lietuvių kalbos atpažinimui iOS įrenginiuose.
2. Ištirti ar šių bibliotekų greitaveika yra pakankama atpažinimui realiu laiku.
3. Ištirti kokio dydžio yra atpažinimo klaidos šiose bibliotekose naudojant tuos pačiuos kalbos modelius.

1.3. Dokumento struktūra

Magistro darbą sudaro: 5 skyriai, literatūros sąrašai, bei priedai.

Pirmajame skyriuje pristatoma darbo tema, darbo tikslas ir uždaviniai.

Antrame skyriuje analizuojamos kalbos atpažinimo panaudojimo sritys, sukurtos sistemos kalbai atpažinti, bei kalbos atpažinime naudojami metodai ir idėjos.

Trečias skyrius yra skirtas paaiškinti principinius kuriamos sistemos iOS platformai ypatumus.

Ketvirtame skyriuje pateikiami sukurtos sistemos eksperimentinio tyrimo rezultatai ir lyginami CMU Sphinx ir Julius bibliotekų rezultatai.

Penktame skyriuje suformuluojamos darbo išvados ir įvardijami planuojami ateities darbai ir perspektyvos.

2. ANALITINĖ DALIS

2.1. Panaudojimo sritys

Automatinio kalbos atpažinimo naudojimo sritys:

- neįgaliesiems;
- transkripcija[1] ir stenografavimas[2];
- vertimas;
- interaktyvios balso atsako sistemos;
- prietaisų valdymas;
- žaidimai.

Neįgaliesiems - tai viena perspektyviausių balso technologijų panaudojimo sričių. Neįgaliems žmonėms turintiems regėjimo sutrikimų ar ribotas judėjimo galimybes balso technologijos yra vienas iš būdų integruotis į visuomenę ir gan dažnai esminis ar net vienintelis. Galima išskirti dvi dideles balso technologijų taikymo neįgaliesiems grupes:

- į informacijos valdymą ar pateikimą balsu orientuoti taikymai, tokie kaip kompiuterio valdymas balsu, informacijos iš interneto ar kitų tekstinių informacijos šaltinių perskaitymas balsu ir pan.;
- neįgaliesiems skirti specializuoti techniniai įrenginiai su integruotomis balso technologijų komponentėmis (pvz, balsu valdomi vežimėliai).

Į neįgaliuosius orientuoti taikymai iš esmės remiasi tais pačiais kalbos apdorojimo ir analizės principais, vienintelis skirtumas yra tas, kad neįgaliems žmonėms priimtinas žemesnis balso technologijų lygis nei eiliniams vartotojams, nes jiems tai dažnai vienintelė alternatyva valdyti įrenginį ar gauti informaciją. Dar viena ypatybė yra ta, kad neįgaliesiems skirtų taikymų panaudojimu rūpinasi ne tiek patys jų vartotojai - neįgalieji, bet valstybinės socialinės rūpybos tarnybos, turtingose šalyse labdaros organizacijos ir pan. [1].

Yra sukurta ir programinė įranga žmonėms turintiems mokymosi sutrikimų „WYNN Reader“, kuri skaito elektroninį tekstą vartotojui, taip pat gali tarti žodžius paraidžiui ir pateikti žodžių apibrėžimus [2].

Transkripcija ir stenografavimas - yra pakankamai aktualus uždavinys kalbant apie kalbos atpažinimą, nes gali būti naudojamas pakankamai įvairiose srityse, kur reikia užrašyti tai kas pasakyta, tokios sritys yra medicina [3], interviu, posėdžiai, forumai, sesijos, derybos, paskaitos, mitingai, konferencijos [4].

Vertimas - yra dar viena sritis, kurioje naudojamas kalbos atpažinimas (bei sintezė). Yra naudojami trys automatinio vertimo metodai:

- taisyklėmis grįstas (angl. *Rule-based*)
- pavyzdžiais grįstas (angl. *Example-based*)
- statistika grįstas (angl. *statistically-based*)

Tačiau dabartinė nuomonė yra, kad geriausi rezultatai gali būti pasiekami apjungus visus tris metodus. Taip pat visi šie trys metodai turi bendrą problemą - jie verčia kalbą tik sakinio ribose be kontekstinės informacijos [5].

Interaktyvios balso atsako sistemos (angl. IRV - interactive voice response) - dabartinės interaktyvios balso atsako sistemos tobulėja ir DTMF (angl. *dual tone multi frequency*) interpretavimą, kai vartotojui paspaudus mygtuką generuojamas dviejų dažnių signalas pagal kurį sistema nustato kokį mygtuką vartotojas paspaudė, keičia balso atpažinimas. Vartotojui norint gauti informaciją jis naviguojamas atpažįstant jo ištartus raktinius žodžius (pvz. sistema paprašo vartotojo pasakyti dieną apie kurią bus pateikiama orų informacija) gan dažnai iš išvardinto sistemos galimų

1 Transkripcija – lingv. tikslus kalbos garsų parašymas pagal tarimą, perraša: Fonetinė transkripcija. (http://www.zodynas.lt/terminu-zodynas/t/transkripcija).

2 Stenografija – stenografija raštas suprastintais ženklais ir sutrumpinimais sakytinei kalbai užrašyti, greitraštis (http://www.zodynas.lt/terminu-zodynas/S/stenografija).

žodžių sąrašo [11].

Prietaisų valdymas - vis daugiau prietaisų yra kuriama su balso sąsaja. Jau seniai niekam nėra naujiena galimybė mobiliajame telefone balsu ištarus adresato vardą jam skambinti. Dabar jau yra sukurti balsu valdomi televizoriai [6], skalbimo mašinos [7], netgi Xbox Kinect priedėlis turi integruotą mikrofoną ir galimybę konsolę kontroliuoti balsu [8].

Žaidimai - kompiuteriniuose žaidimuose taip pat taikomos kalbos atpažinimo technologijos. Yra sukurta programa Say2Play, kuri leidžia balsu tarti komandas kurios interpretuojamos žaidime [9]. Ši programa populiariausiems žaidimams turi komandų šablonus ir užtenka tiesiog pasileisti žaidimą su įjungta programa ir bus galima balso komandomis valdyti meniu ar kt. Taip pat jau pradedami kurti žaidimai, kurie iš kart turi integruotas kalbos atpažinimo priemones, tokio žaidimo pavyzdys yra žaidimas Skyrim, kuris naudoja Xbox Kinect priedėlį [10].

2.2. Esamų sprendimų analizė

Nėra sistemos iOS platformai, kuri atpažintų lietuvių kalbą. Taip pat vos dvi iš esamų sistemų gali atpažinti kalbą be interneto prieigos. Kaip matome iš 2.2.1 lentelės dauguma kalbos atpažinimo sistemų yra grįstos kliento-serverio architektūra[3].

2.2.1 lentelė Esamų sistemų palyginimas

	Atpažįsta lietuvių kalbą	Nereikalinga interneto prieiga	Nemokamas	Galimybė apmokyti
Open Ears	-	+	+	-
iSpeech	-	-	-	-
AT & T Speech SDK	-	-	-	-
Nuance	-	-	-	-
Creed Vocal SKD	-	+	-	-

2.3. Signalo paruošimas

Kalbos atpažinimo sistemos prisitaikymas prie akustinės aplinkos yra labai svarbus, nes realiomis sąlygomis tokie dalykai, kaip akustinis aidas ir pašaliniai aplinkos triukšmai iškraipo kalbos signalą ir labai apsunkina jo apdorojimą kalbos atpažinimo sistemose [12]. Todėl realiomis sąlygomis atpažįstant kalbą yra būtina naudoti akustinio aido slopinimo (angl. *acoustical echo cancellation*) ir triukšmų slopinimo (angl. *noise supression*) metodus, kurie pagerintų sistemos atpažinimo kokybę. Kalbos kokybės (angl. *speech enhancement*) gerinimas yra labai svarbus uždavinys prieš atliekant patį kalbos signalo apdorojimą ir kalbos atpažinimą.

2.3.1. Akustinio aido slopinimas

Akustinio aido slopinimas (angl. *acoustical echo cancelation*) yra svarbus norint pagerinti signalo kokybę. Akustinis aidas gali atsirasti dėl keleto priežasčių: pačios aplinkos nuo kurios atsispindi garsas ir mažų prietaiso tokio kaip GSM telefonas matmenų, kuriame dėl mažo atstumo tarp mikrofono ir garsiakalbio susidaro aidas [13]. Akustinio slopinimo uždavinyje naudojami įvairūs filtrai, kuriais bandoma nufiltruoti aidą, galimi filtrų tipai yra tokie[13]:

- **Tiesiniai filtrai** (angl. *FIR - finite impulse response*)
- **Netiesinės struktūros** (angl. *nonlinear structures*) filtrai, jie gali būti modeliuojami šiais modeliais:
    - Voltera modeliu (angl. *Volterra model*)
    - Bilinear modeliu (angl. *Bilinear model*)

3 Kliento-serverio architektūra – tai architektūra, kur verslo logika yra nutolusiuose serveriuose, o klientinė programinė įranga kreipiasi į serverį prašydama informacijos.

- **Netiesinės kaskadinės struktūros** (angl. *nonlinear cascade structure*) filtrai, pagrindinis jų privalumas mažesnis kiekis parametrų kuriuos reikia apskaičiuoti. Jie gali būti modeliuojami šiais modeliais:
    - Hamerštaino modeliu (angl. *Hammerstein model*)
    - Vynerio modeliu (angl. *Wiener model*)
    - Vynerio-Hamerštaino modeliu (angl. *Wiener-Hammerstein model*)

Kalbos atpažinimo sistemai pateikus signalą, kuris yra su nufiltruotu akustiniu aidu galima labai žymiai sumažinti sistemos žodžių atpažinimo klaidą (angl. *WER - word error rate*) (eksperimentiniais rezultatais nuo 74,5% iki 29,1%) [13].

2.3.2. Triukšmų slopinimas

Triukšmų slopinimas (angl. *noise supression*) yra labai svarbus, natūrali aplinka yra pilna triukšmų ir kalbos atpažinimas, kai kalbos signale yra triukšmo dedamųjų tampa sudėtingesnis ir prastėja atpažinimo tikslumas. Pagrindinis triukšmo šalinimo principas yra charakterizuoti triukšmą tylos intervaluose ir atimti jį iš signalo, o pagrindinė esmė daugumos metodų yra adaptyvūs filtrai (angl. *adaptive filters*) taikomi kalbos spektrui [14]. Žemiau išvardyti taikomi metodai:

- Adaptyvus Vynerio filtravimas (anlg. *Adaptive Wiener Filtering*)
- Adaptyvus linijos stiprininimas (angl. *An Adaptive Line Enchancer*)
- Daugelio mikrofonų technika (angl. *A Multiple Microphone Technique*)
- Spektro atėmimas (angl. *Spectral Substraction*)
- Kepstro vidurkio atėmimas (angl. *Cepstral Mean Subtraction*)
- Aklas stiprinimas (angl. *Blind Equalization*)

Yra net bandymų apjungti kelis triukšmų slopinimo metodus ir naudoti juos kartu, IDIAP institute atliktame tyrime apjungus spektro atėmimo ir kepstro vidurkio metodus arba spektro atėmimo ir aklo slopinimo metodus buvo gauti geri rezultatai. Pastaroji kombinacija yra labiau taikytina realioms atpažinimo sistemoms, nes yra kadrais sinchroninė (angl. *frame synchronous*) [15].

2.4. Melų skalės kepstro koeficientai

Audio signalas neša daug perteklinės informacijos, kuri visiškai nenaudinga kalbos atpažinime. Todėl reikia kažkokiu būdu modeliuoti audio signalą ir išskirti iš jo parametrus, kurie geriau atspindi žmogaus suvokiamą garsą. Yra žinoma, kad žmogus dažnių skalę suvokia ne tiesiniu masteliu, todėl buvo sukurtos skalės tokios kaip barkų, melų ir kt., kurios geriau atspindėjo psichoakustinį žmogaus garso suvokimą. Melų skalė buvo išvesta atliekant eksperimentus su sinusoidiniais signalais ir ji gali būti aproksimuota:

$$B(f)=1125\cdot\ln\left(1+\frac{f}{700}\right) \tag{1}$$

Atvirkštinė transformacija iš melų į hercus:

$$B^{-1}(b)=700\cdot\left(e^{\frac{b}{1125}}-1\right) \tag{2}$$

Melų skalės kepstro koeficientai (MSKK) yra apibrėžiami, kaip realus kepstras, gaunami iš signalo kadro, padauginto iš lango funkcijos ir apskaičiuotos Furjė transformacijos. Jie skiriasi nuo realaus kepstro tuo, kad yra apskaičiuoti panaudojant specialius trikampius filtrus, suformuotus pagal netiesinę dažnių skalę. Apibrėžiamas filtrų rinkinys, susidedantis iš I trikampių filtrų. i -tasis filtras apibrėžiamas:

$$H(i,f)=\begin{cases}0, & kai\ f<f(i-1);\\ 2\dfrac{f-f(i-1)}{(f(i+1)-f(i-1))(f(i)-f(f-1))}, & kai\ f(i-1)\le f\le f(i);\\ 2\dfrac{f(i+1)-k}{((i+1)-f(i-1))(f(i+1)-f(i))}, & kai\ f(i)\le f\le f(i+1);\\ 0, & kai\ f>f(i+1)\end{cases} \tag{3}$$

Šie filtrai skaičiuoja vidurkinį spektrą aplink kiekvieną centrinį dažnį su vis platėjančiomis dažnių juostomis. Ribiniai dažniai $f(i)$ išsidėstę pagal melų skalę. Juos galima išreikšti:

$$f(i)=\left(\frac{N}{F_d}\right)B^{-1}\left(B(F_a)+\frac{i\cdot B(F_v)-B(F_a)}{I+1}\right) \tag{4}$$

Čia N – GFT dydis, I – filtrų skaičius, F_d – diskretizacijos dažnis (hercais), F_a ir F_v – žemiausias ir aukščiausias filtro dažnis (hercais). B^{-1} – duotas formulėje (2).

Energijos logaritmą filtro išėjime (melų skalės spektro koeficientus – MSSK) galima išreikšti:

$$C(i)=\ln\left(\sum_{k=0}^{N-1}H(i,k)\cdot|S(k)|^2\right),\ i=1,2,...,I \tag{5}$$

Čia $S(k)$ – signalo Furjė transformacijos koeficientai (signalo spektro) amplitudės.

Melų skalės kepstrą galima išreikšti pasinaudojus diskrečiaja kosinusų transformacija [Kamarauskas, 2009, 53-55]:

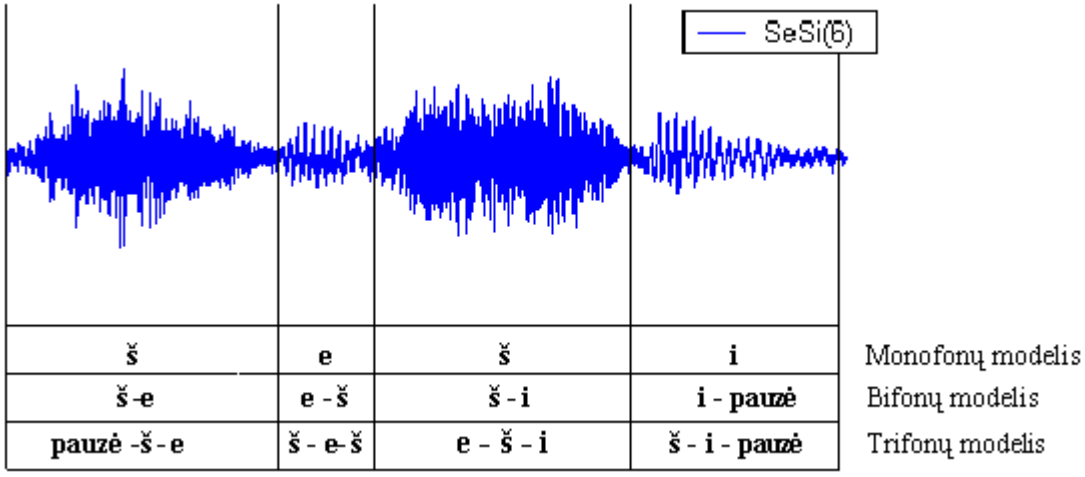
$$c(j)=\sum_{i=0}^{I-1}C(i)\cdot\cos\left(\pi n\frac{i-1}{2I}\right),\ j=0,1,...,I \tag{6}$$

MSKK atspindi garso signalo spektro energijos pasiskirstymą prasmingu atpažinimo sistemai būdu (įprasminant, kaip žmogus suvokia garso signalo spektrą). Dėl šios priežasties MSKK yra labiausiai naudojami parametrai kalbos ir kalbėtojo atpažinime, taip pat ir garso klasifikavime [24].

2.5. Akustiniai modeliai

Svarbus klausimas kalbos atpažinime yra akustinio modelio sudarymas t. y., kaip parinkti akustinio modeliavimo vienetus [21] pagal pagal fonetinius vienetus. Atpažinimo vienetų pasirinkimas yra pirmas ir vienas iš svarbiausių sprendimų. Vienetų pasirinkimas turi savų trūkumų ir privalumų, nes vienetai gali būti skirtingo ilgio (trukmės), kas įtakos atpažinimo rezultatus.

Kuo ilgesnis analizuojamas fragmentas, tuo tiksliau jis atspindi žmogaus generuojamą akustinį kalbos signalą ir tuo tiksliau modeliuoja konteksto įtaką garsų susidarymui. Pavyzdys būtų, kai kiekvienam žodžiui sukuriamas atskiras akustinis modelis, tačiau toks modeliavimas yra ribotas ir tinkamas tik nedidelio žodyno uždaviniams, nes augant žodynui apmokymas tampa sudėtingas, dėl žodžių galūnių kaitos ir kitų faktorių. Tačiau kartais žodžių modeliai naudojami lygiagrečiai su kitais akustiniais modeliais, kaip tarkim trumpi žodžiai atliekantys tam tikras funkcijas, kaip pavyzdžiui ištiktukai, jungtukai ir pan.



2.5.1 pav. Fonemų akustiniai modeliai

Paprastai atpažinimui naudojami vienetai apsiriboja fonemomis (monofonų modelis) ar fonemų junginiais (bifonų, trifonų modelis) (2.5.1 pav.). Šie modeliai naudojami todėl kad kiekvienoje kalboje galimų fonemų skaičius nedidelis lyginant su žodžiais ir iš anksto apibrėžtas, bei baigtinis. Taip pat nėra sudėtinga surasti pakankamai bet kurios fonemos realizacijos pavyzdžių. Modeliuojant kalbos signalą fonemų lygiu kiekvienas žodis segmentuojamas ir transkribuojamas į fonemų sekas. Turint fonemų akustinius modelius žodžiai yra modeliuojami jungiant fonemų sekas. Tačiau atskirų fonemų modeliai turi didelį trūkumą, jie daro prielaidą, kad bet kuri fonema bet kokiame kontekste yra tokia pati, kas yra iš esmės klaidinga. Todėl yra naudojami bifonų ir trifonų modeliai, kurie atsižvelgia į gretimų fonemų kontekstą. Naudojant šiuos modelius atpažinimo tikslumas išauga, tačiau trifonų minusas yra tas, kad tarkim jei kalboje yra 50 fonemų, tai trifonų skirtingų kombinacijų bus $50^3=125000$, o difonų $50^2=2500$. Bandant realizuoti atpažinimą realiame laike, galimų kombinacijų skaičius turi didelę įtaką.

Dauguma šiuolaikinių kalbos atpažinimo sistemų naudoja PMM, kurių būsenos modeliuoja kalbos signalą, kaip tiesinę fonemų seką su papildomais tarnybiniais akustiniais įvykiais (tyla, triukšmas ir pan.)

2.6. Kalbos modelis

Kuo didesnis žodynas, tuo daugiau jame yra panašiai skambančių žodžių ir atpažinimo uždavinys sudėtingėja. Kalbos modelis savo ruožtu apriboja įmanomų žodžių kombinacijas, naudodamas iš didelės apimties teksto sukauptą statistinę informaciją. Dažniausiai naudojamas yra n-gramų [22] metodas, kuris modeliuoja šalia einančius n žodžių (1 – unigramų, 2 – bigramų, 3 – trigramų).

Kuo didesnis žodynas, tuo kalbos modelių naudojimas tampa svarbesnis. Esant nedideliam žodynui ar atliekant atskirų žodžių atpažinimą kalbos modeliai paprastai nėra naudojami.

2.7. Statistiniai metodai

2.7.1. Paslėptas Markovo modelis

PMM panaudojimas kalbai atpažinti remiasi prielaida, kad kalbos signalas yra atsitiktinis procesas, kurio parametrus galima nustatyti. Šiame metode kalbos pavyzdžiai modeliuojami paslėptaisiais Markovo modeliais. PMM modeliuose yra du atsitiktiniai procesai: perėjimas iš vienos būsenos į kitą (Markovo procesas) ir stebėjimas būsenoje [18].

Bus naudojamas žymėjimas [16]:

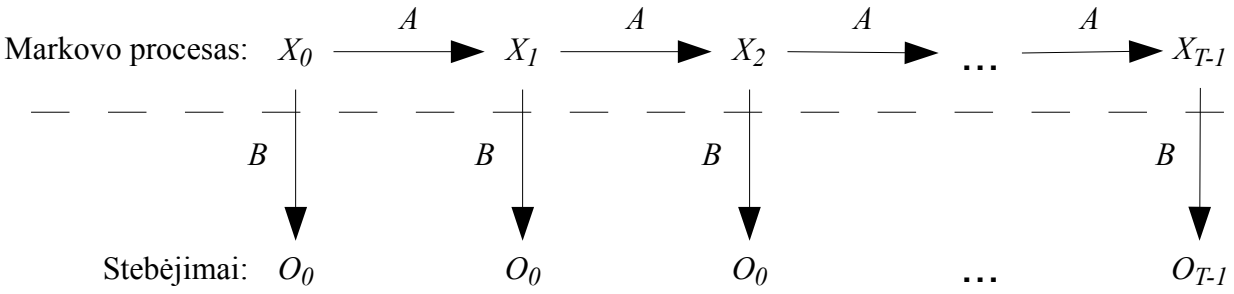
T = stebėjimų skaičius

N = būsenų skaičius

M = stebimų simbolių skaičius

$Q = \{q_0, q_1, \ldots, q_{N-1}\}$ = Markovo proceso atskiros būsenos
$V = \{0,1,\ldots, M-1\}$ = stebėjimų rinkinys
A = būsenų perėjimo tikimybės
B = stebėjimų tikimybių matrica
π = pradinės būsenos pasiskirstymas
$O = (O_0, O_1, \ldots, O_{T-1})$ = stebėjimų seka
Taip pat, $O_i \in V$ kiekvienam $i = 0,1,\ldots, T-1$.

Apibendrintas PMM pavaizduotas 2.7.1.1 pav, jame X_i reiškia paslėptų būsenų seką. Paveiksle pavaizduotas Markovo procesas, kuris ir yra paslėptas, atskirtas brūkšnine linija. Galima stebėti tik O_i stebėjimus, kurie yra tiesiogiai susiję paslėptomis Markovo proceso būsenomis per matricą B.



2.7.1.1 pav. Paslėptas Markovo modelis

Naudojant PMM yra sprendžiami trys uždaviniai [17]:

1. Turint $\lambda = (A, B, \pi)$ ir stebėjimų seką O, rasti $P(O \mid \lambda)$. Norima nustatyti tikimybę, kad turimas modelis generuos stebėjimų O seką.
2. Turint $\lambda = (A, B, \pi)$ ir stebėjimų seką O, rasti optimalią būsenų seką paslėptam Markovo procesui. Kitais žodžiais norima sužinoti paslėptą PMM dalį.
3. Turint stebėjimų seką O ir dimensiją N ir M, rasti modelį $\lambda = (A, B, \pi)$, su kuriuo sekos O tikimybė yra didžiausia. Į šį uždavinį galima žvelgti, kaip į modelio apmokymą geriausiai atspindėti stebėjimų duomenis.

1 uždavinio sprendimas

Tegul $\lambda = (A, B, \pi)$ yra duotas modelis, o $O = (O_0, O_1, \ldots, O_{T-1})$ yra stebėjimų seka. Uždavinys yra rasti $P(O \mid \lambda)$.

Tegul $X = (x_0, x_1, \ldots, x_{T-1})$ būna būsenų seka, tuomet iš B apibrėžimo turime

$$P(O \mid X, \lambda) = b_{x_0}(O_0) b_{x_1}(O_1) \ldots b_{x_{T-1}}(O_{T-1}) \tag{7}$$

ir pagal π ir A apibrėžimus

$$P(X \mid \lambda) = \pi_{x_0} a_{x_0, x_1} a_{x_1, x_2} \ldots a_{x_{T-2}, x_{T-1}} \tag{8}$$

Kadangi

$$P(O, X \mid \lambda) = \frac{P(O \cap X \cap \lambda)}{P(\lambda)} \tag{9}$$

ir

$$P(O \mid X, \lambda) P(X \mid \lambda) = \frac{P(O \cap X \cap \lambda)}{P(X \cap \lambda)} \cdot \frac{P(X \cap \lambda)}{P(\lambda)} = \frac{P(O \cap X \cap \lambda)}{P(\lambda)} \tag{10}$$

turime

$$P(O, X, \mid \lambda) = P(O \mid X, \lambda) P(X \mid \lambda) \tag{11}$$

Sumuodami visas galimas būsenų sekas gauname

$$\begin{aligned} P(O \mid \lambda) &= \sum_X P(O, X \mid \lambda) \\ &= \sum_X P(O \mid X, \lambda) P(X \mid \lambda) \\ &= \sum_X \pi_{x_0} b_{x_0}(O_0) a_{x_0 x_1} b_{x_1}(O_1) \ldots a_{x_{T-2}, x_{T-1}} b_{x_{T-1}}(O_{T-1}) \end{aligned} \tag{12}$$

Vis dėlto, tiesiogiai apskaičiuoti šią išraišką yra neįmanoma, nes tam reikia apie $2TN^T$ sandaugų. PMM stiprybė yra, kad egzistuoja efektyvūs algoritmai, kurių pagalba galima gauti tą patį rezultatą su žymiai mažesniu skaičiavimų kiekiu.

Rasti $P(O \mid \lambda)$ naudojamas pirmyn algoritmas (angl. f*orward algorithm*) dar vadinamas α -perėjimu (angl. α -*pass*). Visiems $t = 0, 1, \ldots, T-1$ ir $i = 0, 1, \ldots, N-1$, apibrėžiame

$$\alpha_t(i) = P(O_0, O_1, \ldots, O_t, x_t = q_i \mid \lambda) \tag{13}$$

Tuomet $\alpha_t(i)$ yra dalinės stebėjimo sekos iki t tikimybė, kurioje Markovo procesas yra būsenoje q_i laiko momentu t.

Esminė įžvalga yra, kad $\alpha_t(i)$ galima apskaičiuoti rekursiškai.

1. Tegul $\alpha_0(i) = \pi_i b_i(O_0)$ visiems $i = 0, 1, \ldots, N-1$
2. Kiekvienam $t = 0, 1, \ldots, T-1$ ir $i = 0, 1, \ldots, N-1$ apskaičiuojame

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} a_{t-1}(j) a_{ji} \right] b_i(O_t) \tag{14}$$

3. Tuomet iš (13)formulės aišku, kad

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} a_{T-1}(i) \tag{15}$$

Pirmyn algoritmui reikia tik apie $N^2 T$ sandaugų, kas yra daug kartų geriau nei $2TN^T$ skaičiuojant tiesiogiai.

2 uždavinio sprendimas

Turint $\lambda = (A, B, \pi)$ ir stebėjimų seką O rasti labiausiai tikėtiną būsenų seką. „Labiausiai tikėtina“ PMM atveju reiškia, kad norima maksimizuoti laukiamą skaičių teisingų būsenų, kai dinaminio programavimo atveju ieškomas daugiausiai taškų surinkęs kelias, kas reiškia, kad PMM ir dinaminio programavimo sprendimai nebūtinai sutaps.

Iš pradžių apibrėžiamas atgal algoritmas (angl. *backward algorithm*) dar vadinamą β -perėjimu (angl. β -*pass*). Šis algoritmas yra analogiškas pirmyn algoritmui aprašytam aukščiau, išskyrus tai, kad atgal algoritmas darbą pradeda nuo pabaigos ir eina link pradžios.

Visiems $t = 0, 1, \ldots, T-1$ ir $i = 0, 1, \ldots, N-1$ apibrėžiame

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_{T-1} \mid x_t = q_i, \lambda) \tag{16}$$

Tuomet $\beta_t(i)$ gali būti apskaičiuojama rekursiškai (ir efektyviai) šiais žingsniais.

1. Tegul $\beta_{T-1}(i)=1$, visiems $i=0,1,\ldots,N-1$
2. Kiekvienam $t=T-2,T-3,\ldots,0$ ir $i=0,1,\ldots,N-1$ apskaičiuojame

$$\beta_t(i)=\sum_{j=0}^{N-1} a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{17}$$

Kiekvienam $t=0,1,\ldots,T-2$ ir $i=0,1,\ldots,N-1$ apibrėžiamas

$$\gamma_t(i)=P(x_t=q_i \mid O,\lambda) \tag{18}$$

Kadangi $\alpha_t(i)$ įvertina aktualią tikimybę iki laiko momento t ir $\beta_t(i)$ įvertiną aktualią tikimybę po laiko momento t ,

$$\gamma_t(i)=\frac{\alpha_t(i)\beta_t(i)}{P(O\mid\lambda)} \tag{19}$$

Primename, kad vardiklis $P(O\mid\lambda)$ yra gaunamas sumuojant $\alpha_{T-1}(i)$ per i . Iš $\gamma_t(i)$ apibrėžimo seka, kad labiausiai tikėtina būsena laiko momentu t yra q_i , kuriai $\gamma_t(i)$ yra maksimali, kur maksimumas yra imamas pagal i .

3 uždavinio sprendimas

Šio uždavinio tikslas pakoreguoti modelio parametrus, kad jie geriausiai atspindėtų stebėjimus. Matricų N ir M dydžiai fiksuoti, bet elementų iš A , B ir π turi būti nustatyti, priklausomai nuo eilės stochastinių sąlygų. Tai kad modelis gali būti efektyviai perskaičiuojamas yra dar vienas nuostabus PMM aspektas.

Kiekvienam $t=0,1,\ldots,T-2$ ir $i,j\in\{0,1,\ldots,N-1\}$ apibrėžiamos „di-grammos“, kaip

$$\gamma_t(i,j)=P(x_t=q_i,x_{t+1}=q_j \mid O,\lambda) \tag{20}$$

Tada $\gamma_t(i,j)$ yra tikimybė būti būsenoje q_i laiko momentu t ir pereiti į būseną q_j laiko momentu $t+1$. „Di-grammas“ galima perrašyti pagal α , β , A ir B taip:

$$\gamma_t(i,j)=\frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O\mid\lambda)} \tag{21}$$

$\gamma_t(i)$ ir $\gamma_t(i,j)$ (arba di-gramma) siejasi tarpusavyje sąryšiu

$$\gamma_t(i)=\sum_{j=0}^{N-1}\gamma(i,j) \tag{22}$$

Turint γ ir di-gramma galima patvirtinti, kad modelis $\lambda=(A,B,\pi)$ gali būti perskaičiuojamas.

1. Kiekvienam $i=0,1,\ldots,N-1$ tegu

$$\pi_i=\gamma_0(i) \tag{23}$$

2. Kiekvienam $i=0,1,\ldots,N-1$ ir $j=0,1,\ldots,N-1$ apskaičiuojame

$$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i,j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i) \tag{24}$$

3. Kiekvienam $j = 0, 1, ..., N-1$ ir $k = 0, 1, ..., M-1$ apskaičiuojame

$$b_j(k) = \sum_{t \in \{0,1,...,T-2\}\ O_t = k} \gamma_t(j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i) \tag{25}$$

Perskaičiuojamo a_{ij} skaitiklis yra laukiamas skaičius perėjimų iš būsenos q_i į būseną q_j, kai tuo atveju vardiklis yra laukiamas perėjimų skaičius iš būsenos q_i į bet kurią kitą būseną. Tuomet šis santykis ir yra perėjimo tikimybė iš būsenos q_i į būseną q_j, kas ir yra norima a_{ij} reikšmė.

Perskaičiuojamo $b_j(k)$ skaitiklis yra skaičius kartų modelis yra būsenoje q_j turint stebėjimą k, kai vardiklis yra laukiamas skaičius modelis yra būsenoje q_j. Šis santykis ir yra tikimybė stebėti k, turint modelį būsenoje q_j, kad ir yra norima $b_j(k)$ reikšmė.

Modelio perskaičiavimas yra iteracinis procesas. Iš pradžių inicijuojamas $\lambda = (A, B, \pi)$ su geriausiu spėjimu arba jei nėra turimas joks protingas spėjimas pasirenkamos atsitiktinės reikšmės tokios, kad $\pi \approx 1/N$ ir $a_{ij} \approx 1/N$ ir $b_j(k) \approx 1/M$. Labai svarbu, kad A, B ir π būtų parinkti atsitiktinai, nes vienodos reikšmės lems lokalų maksimumą iš kurio modelis negalės išeiti. Taip pat kaip visada A, B ir π turi būti eilutėmis stochastiniai.

Šios problemos sprendimas gali būti apibendrintas kaip:

1. Inicijuoti $\lambda = (A, B, \pi)$
2. Apskaičiuoti $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$ ir $\gamma_t(i)$
3. Perskaičiuoti modelį $\lambda = (A, B, \pi)$
4. Jei $P(O \mid \lambda)$ padidėja grįžti į 2 žingsnį.

Žinoma gali būti pageidaujama sustoti jei $P(O \mid \lambda)$ nepadidėja tam tikru iš anksto apibrėžtu dydžiu ir/arba gali būti nustatyta maksimali iteracijų riba po kurios modelio perskaičiavimas stabdomas.

2.7.2. Viterbi algoritmas

Viterbi algoritmas [19] yra dinaminio programavimo algoritmas, kuris leidžia apskaičiuoti labiausiai tikėtiną būsenų seką iš stebėjimų sekos.

Norint rasti labiausiai tikėtiną būsenų seką $Q = \{q_0, q_1, ... q_{T-1}\}$ atitinkančią stebėjimų seką $O = \{O_0, O_1, ... O_{T-1}\}$ apibrėžiame

$$\delta_t(i) = \max_{q_0, q_1, ..., q_{t-1}} P(q_0, q_1, ..., q_t = i, O_0, O_2, ..., O_t \mid \lambda) \tag{26}$$

$\delta_t(i)$ yra didžiausia vieno kelio tikimybė laiko momentu t, atspindinti pirmus t stebėjimų, kurie baigiasi būsenoje q_i. Iš indukcijos apibrėžimo turime

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(O_{t+1}) \tag{27}$$

Norint išgauti būsenų seką reikia sekti argumentus, kurie maksimizuoja (27) kiekvienam t ir j, tai pasiekiama panaudojus sąrašą $\psi_t(j)$. Galutinė procedūra aprašanti žingsnius reikalingus rasti labiausiai tikėtiną būsenų seką atrodo taip:

1. Inicijuojame

$\delta_t(i) = \pi_i b_i(O_0)$, kur $i = 0, 1, \ldots, N-1$ ir $\psi_t(j) = 0$

2. Rekursiškai vykdomas
   $\delta_t(j) = \max_{0 \le i \le N-1} [\delta_{t-1}(i) b_j(O_t)]$, kur $t = 1, 2, \ldots, T-1$ ir $j = 0, 1, \ldots, N-1$
   $\psi_t(j) = \operatorname{argmax}_{0 \le i \le N-1} [\gamma_{t-1}(i) a_{ij}]$, kur $t = 1, 2, \ldots, T-1$ ir $j = 0, 1, \ldots, N-1$
3. Sustojimo sąlygos
   $P^* = \max_{0 \le i \le N-1} [\delta_T(i)]$
   $q_t^* = \operatorname{argmax}_{0 \le i \le N-1} [\delta_T(i)]$
4. Būsenų sekos suradimas
   $q_t^* = \psi_{t+1}(q_{t+1}^*)$, kur $t = T-1, T-2, \ldots, 0$

Dažniausiai praktikoje naudojamas modifikuotas Viterbi algoritmas, kuris naudoja spindulinę paiešką (angl. b*eam search*) [20], kuri paspartina algoritmo darbą sumažindama paieškos erdvę.

2.8. Neuroniniai tinklai

2.8.1. Naudojimas

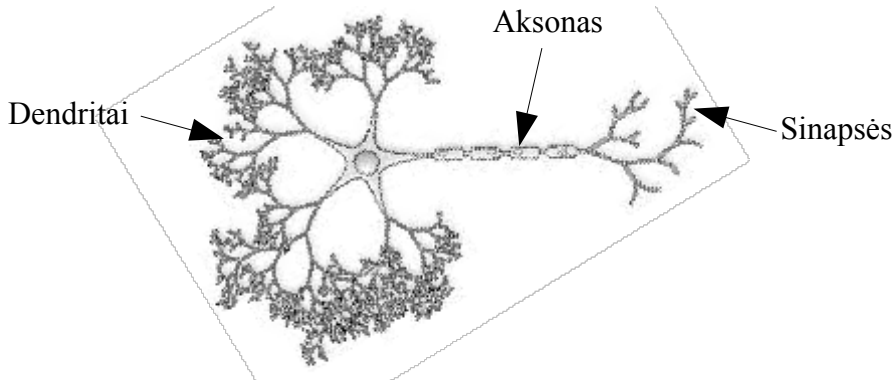
Neuroniai tinklai (angl. *ANN - Artificial Neural Network*) yra žinoma koncepcija nuo šešto dešimtmečio, o bandymai pagerinti kalbos atpažinimo sistemos charakteristikas juos naudojant prasidėjo devintame dešimtmetyje ir parodė žadančių rezultatų didelio žodyno kalbos atpažinime (angl. *large vocabulary speech recognition*), tačiau jie nėra naudojami komercinėse kalbos atpažinimo sistemos, nes CD-GMM-HMMs (angl. *context-dependent Gaussian mixture model HMMs*) pagal greitaveiką (angl. *performance*) juos lenkia. Visgi tobulėjant supratimui apie neuroninius tinklus ir kaip juos apmokyti, ypač apmokant gilius neuroninius tinklus (angl. *DNN - deep neural network*). Galima į juos žvelgti iš naujos perspektyvos juolab, kad neuroninio tinklo apmokymas yra pagrindinis sistemos pagrįstos jais komponentas.

Kalbos atpažinimo sistemos modelis naudoja kalbos garsų fragmentus, pavyzdžiu galima laikyti fonemą, ir paprastai šio modelio žodynas susideda iš 30 ar panašaus skaičiaus tariamų vienetų. Naujausia kalbos atpažinimo technologija naudoja dar mažesnius kalbos fragmentus vadinamus senonais (angl. *senones*), kurių skaičius siekia tūkstančius. Kalbos atpažinimo sistemose panaudojus gilius neuroninsiu tinklus CD-DNN-HMM architektūroje, gauti rezultatai gan ženkliai lenkia iki tol geriausiu laikytą CD-GMM-HMM architektūrą [25].

Nors sistemos naudojančios CD-GMM-HMM tipo architektūrą yra dar tyrimų stadijoje, tačiau jau dabar teikiami žadantys rezultatai gali pastūmėti kalbos atpažinimo technologijas dar arčiau žmogaus lygio atpažinimo tikslumo.

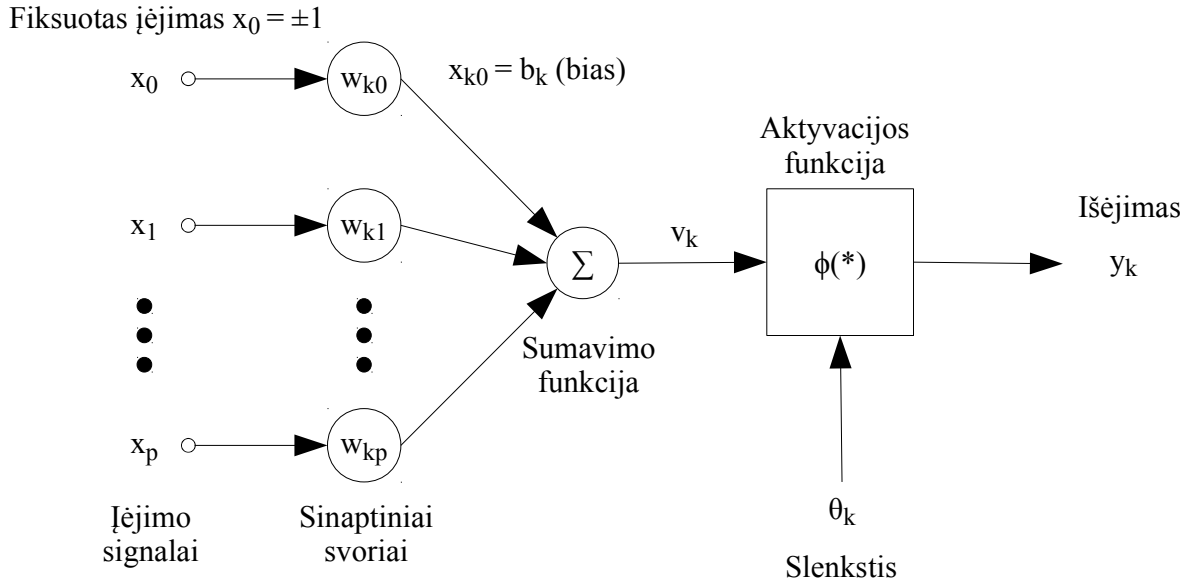
2.8.2. Pagrindinis principas

Neuroniniai tinklai yra įkvėpti neurobiologijos ir pagrindinių žmogaus smegenų veikimo principų. Žmogaus smegenys pasižymi unikalia galimybe mokytis ir sudėtingais sąryšiais susieti objektus. Taigi neuroniniais tinklais yra bandoma šią žmogaus savybe modeliuot. ANN kaip ir smegenys sudaryti iš neuronų (2.8.2.1 pav.).



2.8.2.1 pav. Biologinis neurono modelis

ANN naudojami matematiniai neuronų modeliai [26]. Modeliuojant matematinį neurono modelį svarbu atkreipti dėmesį į tam tikrus dalykus. Pirmiausia biologinio neurono sinapsės yra modeliuojamos kaip svoriai. Sinapsės yra atsakingos už sąryšius tarp neuronų ir už šio sąryšio stiprumą. Neigiamas svoris reiškia slopinantį ryšį, kai tuo atveju teigiami svoriai reiškia stiprinantį ryšį. Kaip matome 2.8.2.2 pav. įėjimo signalai yra modifikuojami priklausomai nuo svorio ir susumuojami. Šis rezultatas toliau yra apdorojamas aktyvacijos funkcijos, kuri kontroliuoja išėjimo signalo amplitudę. Dažniausiai išėjimo signalas yra tarp 0 ir 1, tačiau jis gali būti tarp -1 ir 1.

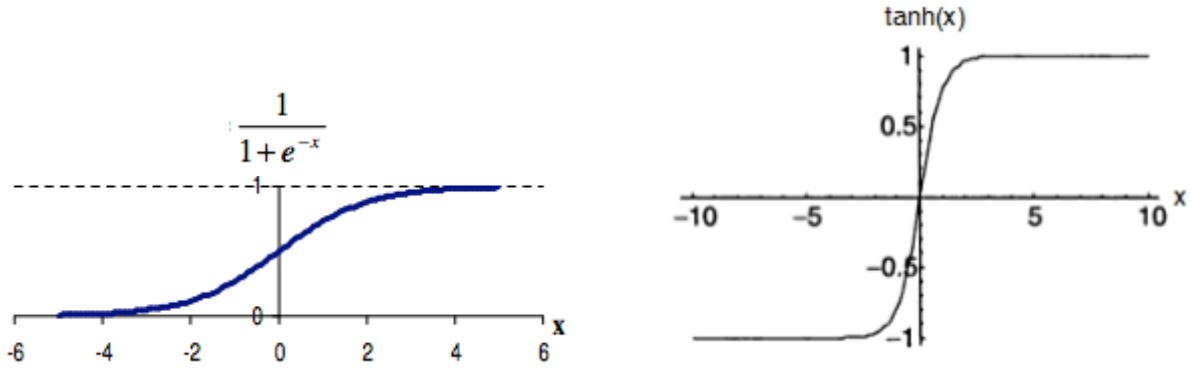


2.8.2.2 pav. Matematinis neurono modelis

Neurono aktyvumas v_k yra apskaičiuojamas pagal formulę:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j \tag{28}$$

Ši v_k reikšmė yra apdorojama aktyvacijos funkcijos, kuri kontroliuoja išėjimo reikšmę y_k. Aktyvacijos funkcijai parinkti galima bet kokia funkciją, kuri v_k reikšmę apibrėžia baigtiniu diapazonu: Bi-polinis sigmoidas (angl. *Bi-polar sigmoid*), Uni-polinis sigmoidas (angl. *Uni-polar sigmoid*), Tanh, Kūginio pjūvio funkcija (angl. *Conic section*), Radialinų bazių funkcija (angl. *Radial Bases Function*) ir kt. Tačiau realiems taikymams geriausiai tinka Uni-polinis sigmoidas ir hiperbolinis tangentas [27].



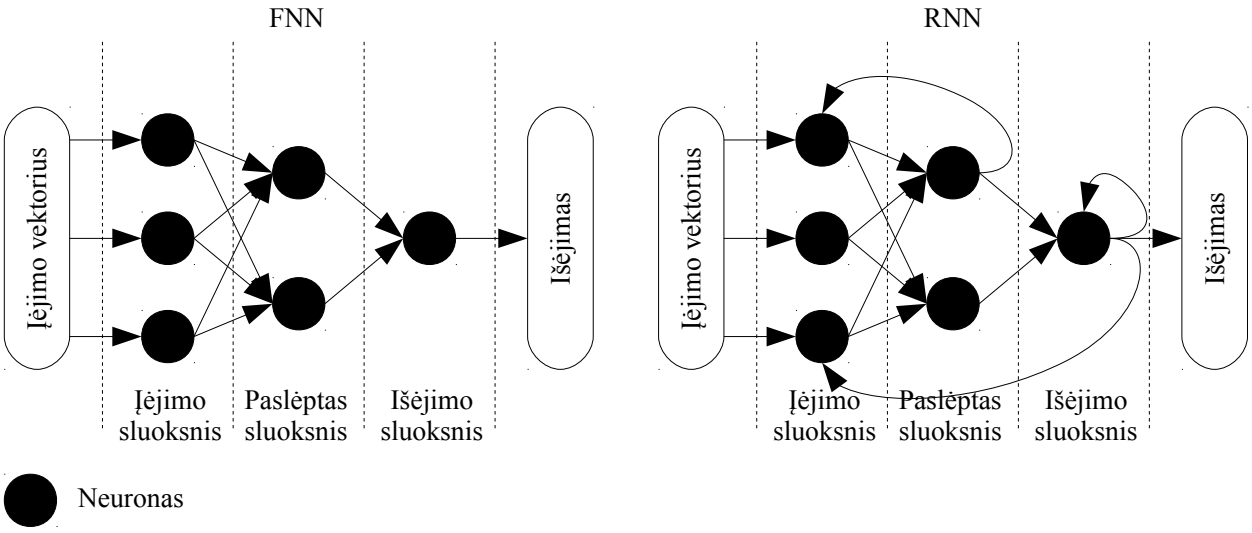
Uni-polinis sigmoidas Hyperbolinis tangentas

2.8.2.3 pav. Aktyvacijos funkcijų pavyzdžiai

Slenkstinė funkcija apibrėžia kraštines v_k reikšmes, kurioms esant išėjime bus 0 arba 1. Turėti slenkstinę funkciją yra naudinga, nes atlikus paprastą palyginimo operaciją kraštinių reikšmių atveju galima išvengti bereikalingo aktyvacijos funkcijos vykdymo, kuri skaičiavimo resursų aspektu yra žymiai sudėtingesnė.

2.8.3. Tinklų topologijos

Matematinis neurono modelis yra įdomus, bet niekuo neypatingas, kol nepradedama jų jungti į neuroninius tinklus. Tada ir atsiskleidžia tikrasis jų potencialas ir skaičiavimų galia. Neuronų jungimas tarpusavyje yra vadinamas topologija ir skirstomas į dvi pagrindines grupes: FNN (angl. *Feed-forward neural network*) – be grįžtamųjų ryšių ir RNN (angl. *Recurent neural network*) – su grįžtamaisiais ryšiais 2.8.3.1 pav.



2.8.3.1 pav. Neuroninio tinklo be grįžtamųjų ryšių (FNN) ir su grįžtamaisiais ryšiais (RNN) topologijos

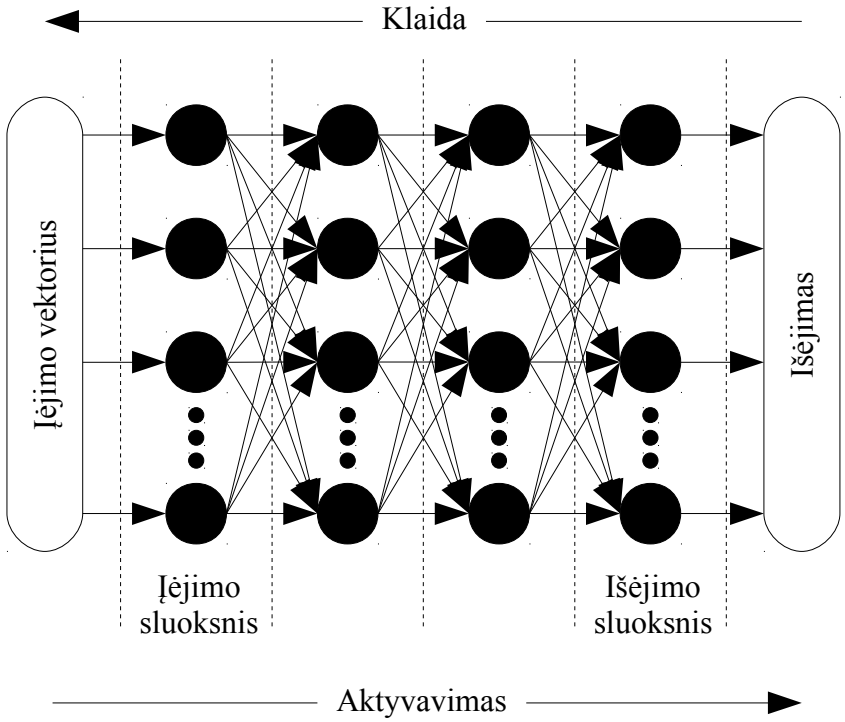
FNN neuroniniai tinklams pagrindinė sąlyga yra, kad informacija privalo keliauti nuo įėjimo link išėjimo ir negali būti jokių grįžtamųjų ryšių. Šie tinklai gali turėti neribotą skaičių sluoksnių, bet kokio tipo aktyvacijos funkcijas ir nerinotą skaičių sąryšių.

RNN neuroniniai tinklai pasižymi tuo, kad gali turėti neribotą skaičių grįžtamųjų ryšių, taip pat kaip ir FNN neribotą skaičių sluoksnių ir t.t. Yra specifinių neuroninių tinklų su grįžtamaisiais ryšiais: pilnai rekurentinis (angl. *Fully recurrent*), Hopfield'o, Elman'o, Jordan'o, dvikryptis (angl. *bi-directional*) ir kt. specialūs atvejai [28].

2.8.4. Apmokymas

Kad neuroninis tinklas galėtų spręsti norimą uždavinį, jį reikia apmokyti. Yra trys pagrindinės apmokymo strategijos: apmokymas su mokytoju (anlg. *Supervised learning*), apmokymas be mokytojo (angl. *Unsupervised learning*) ir stiprinamasis apmokymas (angl. *Reinforcement learning*).

Apmokymas su mokytoju (anlg. *Supervised learning*) iš esmės sprendžia klasifikavimo problemą, nes turimas duomenų rinkinys ir žinoma kokį rezultatą duoda kiekvienas šio rinkinio elementas. Tai žinant norima apmokyti neuroninį tinklą, kad jis galėtų klasifikuoti naujus ir dar nematytus duomenis. FNN tinklams dažniausiai apmokymas atliekamas naudojant klaidos skleidimo atgal algoritmą (angl. *Backpropagation algorithm*). Šis algoritmas minimizuoti klaidą išėjime modifikuodamas svorių w_{kp} reikšmes ir veikia naudodamas gradientinio nusileidimo (angl. *Gradient descent*) strategiją.



2.8.4.1 pav. Neuroninio tinklo klaidos skleidimas atgal

Šis algoritmas sudarytas iš dviejų žingsnių:

1. įėjimo reikšmių „skleidimo pirmyn“ link išėjimo
2. paklaidos „skleidimo atgal“ iš išėjimų į įėjimus.

Kiekvieno j -tojo neurono išėjimo reikšmė y_j l -ajame sluoksnyje yra apskaičiuojama taip:

$$y_j = f(a_j) = f\left(\sum_{k=0}^{n_{l-1}} w_{jk}\, y_k\right), \qquad j = 1, \ldots, n_l \tag{29}$$

Klaidos skleidimo atgal algoritmo klaida yra apskaičiuojama pagal formulę:

$$E(W) = \frac{1}{2}\sum_{i=1}^{d} (y_i - t_i)^2 \tag{30}$$

Svoriai pakeičiami pagal formulę:

$$\Delta w_{jk}(t) = -\eta \frac{\partial E_i}{\partial w_{jk}} \tag{31}$$

Dalinės išvestinės išreiškiamos taip:

$$\frac{\partial E_i}{\partial w_{jl} = k} = \frac{\partial E_i}{\partial a_{ij}} = \frac{\partial a_{ij}}{\partial w_{jk}} \tag{32}$$

Iš (29) formulės gauname

$$\frac{\partial a_{ij}}{\partial w_{jk}} = y_{ik} \tag{33}$$

Tegul

$$\delta_{ij} = \frac{\partial E_i}{\partial a_{ij}} \tag{34}$$

Įstačius (34) ir (33) į (32) ir (31) gaunama

$$\frac{\partial E_i}{\partial w_{jk}} = \delta_{ij} y_{ik} \tag{35}$$

ir

$$\Delta w_{jk}^{i} = -\eta \delta_{ij} y_{ik} \tag{36}$$

Čia j -asis neuronas priklauso l -ajam sluoksniui, k -asis neuronas priklauso $(l-1)$ -ajam sluoksniui.

Išėjimų sluoksnyje

$$\delta_{ij} = \frac{\partial E_i}{\partial a_{ij}} = f'(a_{ij})(y_{ij} - t_{ij}) \tag{37}$$

Čia j -asis neuronas priklauso išėjimų sluoksniui L .

Bus randamas $\frac{\partial E_i}{\partial a_{ij}}$ paslėptiesiems neuronams, t. y. Kai j -asis neuronas priklauso l -ajam sluoksniui ir $l < L$. Naudojantis dalinėmis išvestinėmis, bendruoju atveju galima parašyti

$$\delta_{ij} = \frac{\partial E_i}{\partial a_{ij}} = \sum_{s=1}^{n_{l+1}} \frac{\partial E_i}{\partial a_{is}} \cdot \frac{\partial a_{is}}{\partial a_{ij}} \tag{38}$$

Čia n_{l+1} žymi neuronų $(l+1)$ -ajame sluoksnyje skaičių. Išraiška $\frac{\partial E_i}{\partial a_{is}}$ yra lygi dydžiui δ_{is} , apibrėžtam s -ajam neuronui $(l+1)$ -ajame sluoksnyje. Atsižvelgiant į (29) formulę, gauname

$$\frac{\partial a_{is}}{\partial a_{ij}} = f'(a_{ij}) w_{is} \tag{39}$$

Tada paslėptųjų j -ųjų neuronų

$$\delta_{ij} = f'(a_{ij}) \sum_{s=1}^{n_{l+1}} w_{is} \delta_{is} \tag{40}$$

Čia j -asis neuronas priklauso sluoksniui $l < L$, s -asis neuronas priklauso sluoksniui $(l+1)$.

Iš pradžių reikia apskaičiuoti δ_{ij} išėjimų sluoksnyje L pagal (37) formulę. Tada palaipsniui skaičiuoti δ_{ij} paslėptiems neuronams tarpiniuose sluoksniuose $l < L$ naudojantis $(l+1)$-ųjų sluoksnių δ_{ij} reikšmėmis, gautomis pagal (40) formulę [30].

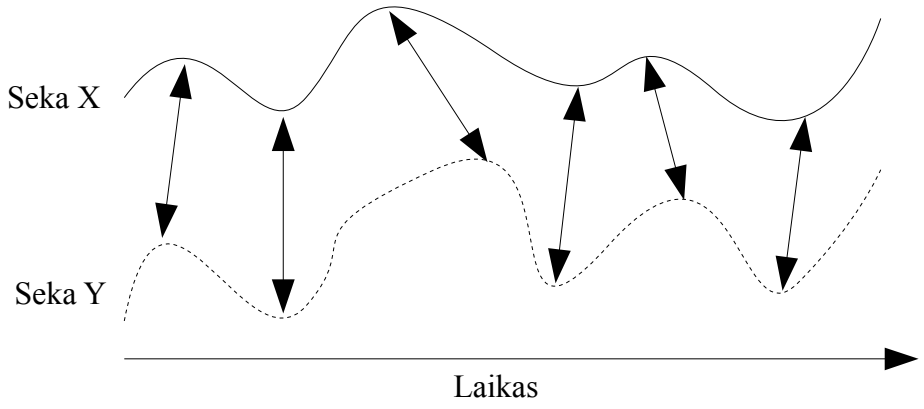
Apmokymas be mokytojo (angl. *Unsupervised learning*) pasižymi tuo, kad turimas nesužymėtų duomenų rinkinys ir išlaidų funkcija (angl. *Cost function*), kurią reikia minimizuoti. Iš esmės naudojantis šiuo apmokymu norima nustatyti paslėptą duomenų struktūra. Apmokyme be mokytojo stengiamės sugrupuoti duomenis į grupes pagal jų panašumą, tačiau kadangi duomenys nėra sužymėti pats neuroninis tinklas stengiasi nustatyti požymius pagal kuriuos šie duomenys turi būti sugrupuoti. Apmokymas be mokytojo daugiausia naudojamas spręsti įvertinimo (angl. *estimation*) problemas tokias kaip: statistinis modeliavimas (angl. *statistical modelling*), suspaudimas (angl. *compression*), filtravimas (angl. *filtering*), aklas šaltinio atskyrimas (angl. *blind source separation*) ir grupavimas (angl. c*lustering*) [29].

Stiprinamasis apmokymas (angl. *Reinforcement learning*) pasižymi tuo, kad apmokymo duomenų paprastai neturi, o neuroninio tinklo parametrai yra apskaičiuojami sąveikaujant su aplinka. Pagrindinis šio apmokymo tikslas yra imtis veiksmų aplinkoje, kad būtų pasiektas tam tikras ilgalaikis apdovanojimas. Dažnai šis apmokymas naudojamas, kaip dalis bendro neuroninio tinklo apmokymo algoritmo.

Kai apibrėžta grąžos funkcija (angl. *Return function*) kuri turi būti maksimizuota, stiprinamasis apmokymas naudoja keletą algoritmų, kad nustatytų strategiją (angl. *policy*), kuri duoda geriausia rezultatą. Pats paprasčiausias naivus brutalios jėgos algoritmas (angl. *Naive brute forve algorithm*) pirmame žingsnyje apskaičiuoja grąžos funkcijos reikšmę kiekvienai strategijai ir pasirenka tą kuri grąžina didžiausią reikšmę. Akivaizdus šio algoritmo trūkumas yra didelis, o kartais ir begalinis galimų strategijų skaičius. Ši problema gali būti apeinama naudojantis vertės funkcijos strategijas (angl. *Value function approaches*) arba tiesiginį strategijos įvertinimą (angl. *Direct policy estimation*). Stiprinamasis apmokymas ypač gerai tinka problemoms, kurios yra koncentruotos į ilgalaikius tikslus, o ne į trumpalaikius [29].

2.9. Dinaminis laiko skalės kraipymas

Dinaminis laiko skalės kraipymas – DLSK [31] (an gl. *Dynamic time warping*) yra gerai žinoma technika skirta rasti optimalų panašumą tarp dviejų (nuo laiko priklausomų) sekų priklausomai nuo tam tikrų apribojimų. Intuityviai sekos yra iškreipiamos netiesiškai, kad atitiktų viena kitą 2.9.1 pav.



2.9.1 pav. Dviejų nuo laiko priklausomų sekų sutapatinimas. Sutapatinti taškai pažymėti rodyklėmis

DLSK tikslas yra palyginti dvi (nuo laiko priklausomas) sekas $X:=(x_1, x_2, ..., x_N)$, kurios ilgis $N\in\mathbb{N}$ ir $Y:=(y_1, y_2, ..., y_M)$, kurios ilgis $M\in\mathbb{N}$. Šios sekos gali būti diskretūs signalai (laiko eilutės (angl. *Time-series*)) arba parametrų (angl., f*eature*) sekos pamatuotos vienodais laiko intervalais. Apibrėžiama parametrų reikšmių sritis F, tada $x_n, y_m\in F$, kiekvienam $n\in[1:N]$ ir $m\in[1:M]$. Kad būtų palygintos dvi skirtingos parametrų sekos $x, y\in F$, apibrėžiamas lokalus skirtingumo įvertis (angl., *local cost measure*) dar kartais vadinamas lokaliu atstumo įverčiu (angl., *local distance measure*), kuris apibrėžiamas kaip funkcija

$$c: F\times F\to\mathbb{R}_{\geq 0} \tag{41}$$

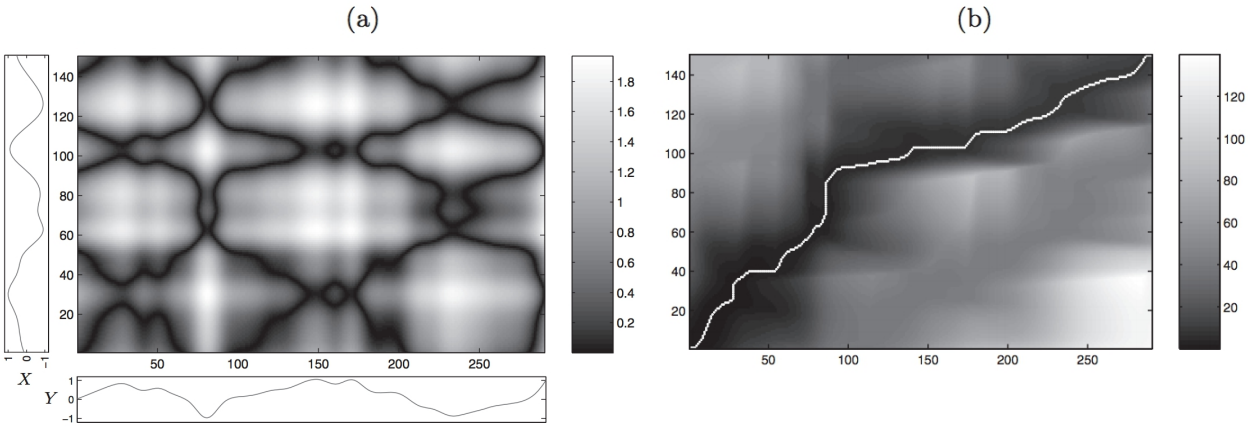
Paprastai $c(x, y)$ yra mažas (mažas įvertis) jei x ir y yra panašūs ir priešingai, jei $c(x, y)$ yra didelis, tai reiškia didelį skirtingumo įvertį. Apskaičiuojant lokalų skirtingumo įvertį kiekvienam sekos X ir Y elementui sukuriama skirtingumo matrica (angl., *cost matric*) $C\in\mathbb{R}^{N\times M}$ apibrėžta $C(n, m):=c(x_n, y_m)$. Tuomet tikslas yra sutapatinti X ir Y su kuo mažesniu skirtingumo įverčiu. Intuityviai optimalus sutapimas (angl., *alignment*) eina per mažo skirtingumo įverčių „slėnį“ matricoje C (3.3.1 pav.).

Optimalus (N, M) sutapatinimo kelias (angl., w*arping path*) yra seka $p=(p_1, ..., p_L)$ su $p_l=(n_l, m_l)\in[1:N]\times[1:M]$, kiekvienam $l\in[1:L]$, kai išpildomos šios trys sąlygos:

(i) Kraštinės sąlygos: $p_1=(1, 1)$ ir $p_L=(N, M)$.

(ii) Monotoniškumo sąlyga: $n_1\leq n_2\leq ...\leq n_L$ ir $m_1\leq m_2\leq ...\leq m_L$.

(iii) Žingsnio dydžio sąlyga: $p_{l+1}-p_l\in\{(1, 0), (0, 1), (1, 1)\}$, kiekvienam $l\in[1:L-1]$.



2.9.2 pav. (a) Skirtingumo įverčių matrica C (b) sukauptų skirtingumo įverčių matrica D su optimaliu sutapatinimo keliu p* (balta linija)

(N, M) sutapatinimo kelias $p=(p_1, ..., p_L)$ apibrėžia sutapimą tarp dviejų sekų $X:=(x_1, x_2, ..., x_N)$ ir $Y:=(y_1, y_2, ..., y_M)$ priskirdamas elementą x_{nl} iš sekos X elementui y_{ml} iš Y. Kraštinės sąlygos užtikrina, kad pirmas ir paskutinis sekų X ir Y elementas yra sutapatintas t. y., sutapatinimas užtikrinamas pilnai sekai. Monotoniškumo sąlyga atspindi tai, kad jei sekantis elementas iš X eina po prieš tai buvusio, tai tas pats galioja ir elementams iš Y ir atvirkščiai. Galiausiai žingsnio dydžio sąlyga užtikrina tęstinumą t. y., kad nei vienas elementas iš X ir Y nėra praleidžiamas ir kad nėra pasikartojančių X ir Y porų.

Bendras skirtingumo įvertis (angl., *total cost*) $c_p(X, Y)$ sutapatinimo keliui p tarp X ir Y atsižvelgiant į lokalų skirtingumo įvertį c yra apibrėžiamas kaip

$$c_p(X, Y):=\sum_{l=1}^{L} c(x_{nl}, y_{ml}) \tag{42}$$

Be to, optimalus sutapatinimo kelias (angl., *optimal warping path*) tarp X ir Y yra sutapatinimo kelias p^* turintis minimalų bendrą skirtingumo įvertį tarp visų sutapatinimo kelių. $DLSK$ atstumas $DLSK(X, Y)$ tarp X ir Y tuomet yra apibrėžiamas kaip bendras skirtingumo įvertis sutapatinimo keliui p^*.

$$\begin{aligned} DLSK(X, Y) &:= c_{p*}(X, Y) \\ &= min\{c_p(X, Y) \mid p\, yra\, (N, M)\, sutapatinimo\, kelias\} \end{aligned} \tag{43}$$

Reikia paminėti, kad $DLSK$ atstumas yra gerai apibrėžtas, nepaisant to, kad gali egzistuoti keli keliai turintys minimalų bendrą skirtingumo įvertį. Taip pat galima pastebėti, kad $DLSK$ atstumas yra simetriškas tuo atveju, kai lokalus skirtingumo įvertis c yra simetrinis. Tačiau $DLSK$ atstumas bendru atveju nėra visada apibrėžtas nepaisant jo priklausomybės nuo c. Be to $DLSK$ atstumas bendru atveju nepatenkina trikampio nelygybės net ir tuo atveju, kai c yra simetrinis.

Norint apskaičiuoti optimalų sutapatinimo kelią p^* galima išbandyti kiekvieną įmanomą kelią tarp X ir Y, tačiau tokio skaičiavimo sudėtingumas auga eksponentiškai nuo N ir M. Todėl naudojamas $O(N M)$ sudėtingumo algoritmas grįstas dinaminiu programavimu. Apibrėžiama dalinė seka $X(1:n) := (x_1, \ldots, x_n)$, kiekvienam $n \in [1:N]$ ir $Y(1:m) := (y_1, \ldots, y_m)$, kiekvienam $m \in [1:M]$ ir rinkinys

$$D(n, m) := DLSK(X(1:n), Y(1:m)) \tag{44}$$

$D(n, m)$ reikšmės apibrėžia $N \times M$ matricą D, kuri vadinama sukauptų skirtingumo įverčių matrica (angl., *accumulated cost matrix*). Akivaizdžiai teisinga tokia lygybė $D(N, M) = DLSK(X, Y)$. Toliau parodysime, kaip efektyviai galima apskaičiuoti D.

Sukauptų skirtingumo įverčių matrica D tenkina tapatybes: $D(n,1) = \sum_{k=1}^{n} c(x_k, y_1)$, kiekvienam $n \in [1:N]$, $D(1,m) = \sum_{k=1}^{m} c(x_1, y_k)$, kiekvienam $m \in [1:M]$ ir

$$D(n, m) = min\{D(n-1, m-1), D(n-1, m), D(n, m-1)\} + c(x_n, y_m) \tag{45}$$

kiekvienam $1 < n \le N$ ir $1 < m \le M$. Taigi, $DLSK(X, Y) = D(N, M)$ yra apskaičiuojamas per $O(N M)$.

Matome, kad (45) išraiška gali būti apskaičiuojama rekursiškai, o inicializacija gali būti supaprastinta išplečiant matricą D papildomu stulpeliu ir eilute ir formaliai priskiriant $D(n,) := \infty$, kiekvienam $n \in [1:N]$ ir $D(0, m) := \infty$, kiekvienam $m \in [1:M]$ ir $D(0,0) := 0$. Tuomet rekursija galioja $n \subset [1:N]$ ir $m \subset [1:M]$. Be to, D gali būti apskaičiuojamas stulpeliais (angl., *column-wise*), kai m-tajam stulpeliui reikia tik $(m-1)$-ojo stulpelio. Tai reiškia, kad jei mus domina tik $DLSK(X, Y) = D(N, M)$ reikšmė, tai šiai reikšmei apskaičiuoti reikia $O(N)$ atminties. Taip pat galima skaičiavimus atlikti eilutėmis (angl., *row-wise*), kam reikės $O(M)$ atminties. Pažymėtina, kad abiem atvejais skaičiavimų laikas yra $O(N M)$. Be to norint rasti optimalų sutapatinimo kelią p^* reikia visos $N \times M$ matricos D. Žemiau pateikiamas algoritmas, kaip šis optimalus sutapatinimo kelias gali būti apskaičiuojamas.

Algoritmas: Optimalus sutapatinimo kelias.
Duomenys: Sukauptų skirtingumo įverčių matrica D.
Rezultatas: Optimalus sutapatinimo kelias p^*.
Procedūra: Optimalus sutapatinimo kelias $p^*=(p_1,...,p_L)$ apskaičiuojamas atvirkštine tvarka pradedant nuo $p_L=(N,M)$. Tarkime apskaičiuojamas $p_l=(n,m)$. Tuo atveju, kai $(n,m)=(1,1)$ turime, kad $l=1$ ir procedūra baigiama. Kitu atveju

$$p_{l-1}:=\begin{cases}(1,m-1), & kai\ \ n=1\\ (n-1,1), & kai\ \ m=1\\ argmin\{D(n-1,m-1),D(n-1,m),D(n,m-1)\}, & kitu\ \ atveju\end{cases} \quad (46)$$

Kaip matome iš 3.3.1 pav. optimalus sutapatinimo kelias p^* eina tik per tas C reikšmes kurių skirtingumo įvertis yra mažas [31].

2.10. Apibendrinimas ir išvados

1. Kalbos atžinimas gali būti panaudotas daugelyje sričių, tačiau tik nedidelio žodyno (pvz. balso komandų) sistemos duoda aukštą atpažinimo tikslumą, kai tuo tarpu stenografavimo sistemos tikslumas būtų varganas ir menkai tinkamas realiam naudojimui, nes tokia sistema turėtų atpažinti rišlią bet kokio konteksto kalbą, kai tuo tarpu šiuolaikinės nuo konteksto priklausomos sistemos tikslumas siekia 62.9% [34].
2. Natūraliomis sąlygomis kalbos signalas yra pilnas pašalinių aplinkos garsų, kurie apsunkina kalbos atpažinimo uždavinį, todėl norint pagerinti atpažinimo tikslumą yra tikslinga naudoti akustinio aido bei triukšmų slopinimą, kas pagerina sistemos tikslumą.
3. Kalbos atpažinime signalas turi būti parametrizuojamas, kad būtų pašalinta perteklinė informacija. Pagrindinis naudojamas metodas signalo parametrizavimui yra melų skalės kepstro koeficientai.
4. Akustinio modelio sudarymas yra labai svarbus žingsnis kalbos atpažinime. Paprastai akustinis vienetas šiuose modeliuose yra fonema, o norint didesnio tikslumo fonemų junginiai. Fonemos naudojamos todėl, kad jų kiekis kalboje yra baigtinis ir mažas lyginant su žodžiais.
5. Augant žodynui tampa aktualu naudoti kalbos modelį, kuris gali sumažinti galimų žodžių kombinacijų kiekį atmesdamas mažai tikėtinas.
6. Kalbos atpažinimui daugiausia naudojami trys pagrindiniai metodai: Markovo modeliai, neuroniniai tinklai ir dinaminis laiko skalės kraipymas. Dinaminis laiko skalės kraipymas gerai tinka vieno kalbėtojo mažo žodyno pavienių žodžių atpažinimui, kai tuo tarpu Markovo modeliais ar neurniniais tinklais grįstos sistemos naudojamos tiek vieno, tiek ir daugelio kalbėtojų kalbos atpažinimui, be to žodyno dydis neapsiriboja mažu žodynu.
7. Daugelis komercinių ir atviro kodo kalbos atpažinimo sistemų yra grįstos Markovo modeliais, nes šis statistinis metodas yra gerai ištyrinėtas, naudojamas ne vieną dešimtmetį ir juo galima pasiekti gerų rezultatų. Kai tuo tarpu neuroniai tinklai dar nėra iki pilnai ištyrinėti ir dar ne visos jų galimybės atskleistos (ypač kas susiję su apmokymu). Be to neuroniniams tinklams reikalingas apmokymo duomenų kiekis paprastai yra didesnis. Dar viena priežastis dėl kurios neuroniniai tinklai yra mažiau naudojami yra ta, kad Markovo modeliais grįstos sistemos šiai dienai lenkia neuroninius tinklus atpažinimo tikslumu, tačiau naujausi tyrimo rezultatai susiję su giliais (angl., *deep*) neuroniniais tinklais duoda daug žadančių rezultatų.
8. Išanalizavus kalbos atpažinimo sistemas skirtas iOS platformai paaiškėjo, kad nėra lietuvių kalbai dedikuotos sistemos ir pagrindinė atpažįstama kalba yra anglų, kartais dar keletas kitų (prancūzų, vokiečių, ispanų ir kt.). Taip pat daliai sistemų reikalinga interneto prieiga, kai iOS

įrenginiai (ypač nauji) turi pakankamai skaičiavimo resursų, kad skaičiavimai galėtų būti atliekami įrenginyje. Ir galiausiai tik viena sistema yra nemokama. Taigi lietuvių kalbą atpažįstanti sistema iOS įrenginiams neturi analogų.

3. PROGRAMINĖS ĮRANGOS PROJEKTAVIMAS

3.1. Planuojamas sprendimas

Bus sukurta kalbos atpažinimo sistema pritaikyta lietuvių kalbos atpažinimui iOS įrenginiuose, kuriai nebus reikalinga interneto prieiga. Šio modulio kūrimui bus naudojamos atviro kodo kalbos atpažinimo bibliotekos CMU Sphinx ir Julius. Šios bibliotekos nėra iš kart paruoštos lietuvių kalbos atpažinimui, todėl reikės susikurti lietuvių kalbos akustinius, žodyno, bei kalbos modelius. Taip pat tik CMU Sphinx biblioteka yra pritaikyta iOS platformai, o Julius biblioteką reikės adaptuoti veikimui iOS įrenginiuose.

Akustinių modelių kūrimą planuojama, taip pat perkelti ant iOS platformos, Tačiau nei viena biblioteka neturi apmokymo modulio pritaikyto iOS platformai, be to Julius biblioteka iš vis neturi apmokymo modulio, todėl reikės rasti sprendimą, kuris tiktų abiem bibliotekoms.

3.2. Naudojamų kalbos atpažinimo įrankių apžvalga

3.2.1. CMU Sphinx

CMU Sphinx yra kalbos atpažinimo sistemų rinkinys susidedantis iš:

- Pocketsphinx – atpažinimo biblioteka parašyta C kalba.
- Sphinxtrain – akustinių modelių apmokymo įrankis.
- Sphinxbase – palaikymo bibliotekos reikalingos Pocketsphinx ir Sphinxtrain.
- Sphinx4 – atpažinimo biblioteka parašyta Java kalba
- CMUcmltk – kalbos modelių įrankių rinkinio.

Iš CMU Sphinx bus naudojamos Pocketsphinx ir Sphinxbase bibliotekos, kurios bus kompiliuojamos ir leidžiamos iOS platformoje, taip pat Sphinxtrain ir Sphinxbase, kurių pagalba bus kuriami akustiniai modeliai.

3.2.2. Julius

Julius yra atpažinimo biblioteka skirta nepertraukiamos (angl., *continuous*) kalbos didelio žodyno kalbos atpažinimu užsiimantiems mokslininkams ir programuotojams. Ši biblioteka yra parašyta C kalba ir optimizuota greitam kalbos atpažinimui ir kaip rašoma, Julius tinklapyje gali atlikti beveik realaus laiko kalbos signalo dekodavimą šiuolaikiniame kompiuteryje, kalbai kurios žodynas siekia 60000 žodžių. Taip pat Julius tinklapyje minima, kad ši biblioteka mažai atminties (iki 64 Mb), kad yra aktualu mobiliuose įrenginiuose. Julius biblioteka bus kompiliuojama ir naudojama iOS platformoje kalbai atpažinti.

3.2.3. HTK Toolkit

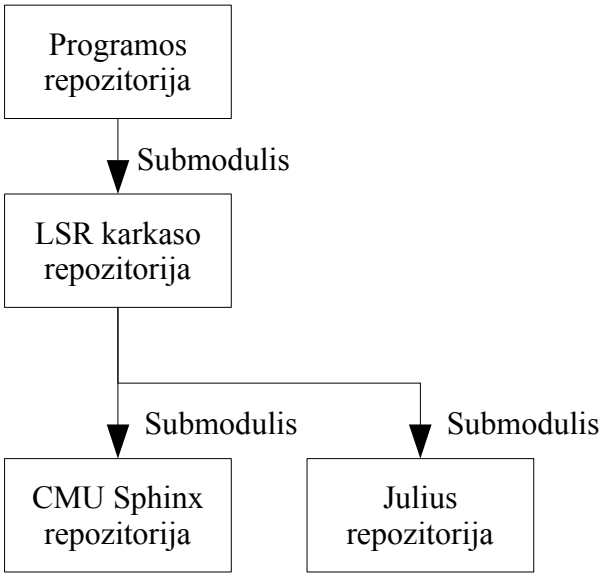
HTK Toolkit yra įrankis skirtas manipuliuoti paslėptais Markovo modeliais ir susidedantis iš atskirų C kalba parašytų bibliotekų modulių. Iš HTK bus naudojamas PMM apmokymo modulis, kad būtų gauti akustiniai modeliai, kurie bus naudojami Julius bibliotekai, nes pastaroji neturi apmokymo įrankių.

3.3. Versijų kontrolė

Sistemos versijavimui bus naudojama Git versijų kontrolę. Kadangi CMU Sphinx ir Julius bibliotekos nėra pilnai paruoštos veikti iOS platformoje gali tekti šias bibliotekas modifikuoti taip, kad jos veiktų, todėl kiekvienai iš šių bibliotekų bus sukurta po atskirą repozitoriją. Taip bus galima sekti kodo pakeitimus ir reikalui esant atnaujinti naudojamas bibliotekas į naujesnes versijas be problemų, kas priešingu atveju padarius pakeitimų bibliotekose būtų pakankamai sudėtinga.

Pačiai lietuvių kalbos atpažinimo sistemai taip pat bus sukurta repozitorija. Kaip atrodys pilna

repozitorijų schema pavaizduota 3.3.1 pav. Kaip matome iš paveikslo kiekviena repozitorija turi jai priklausančius submodulius, kas reiškia, kad pasiimant tarkim programos repozitorija ar lietuvių kalbos atpažinimo repozitoriją automatiškai bus gaunamos ir kitos reikalingos repozitorijos.

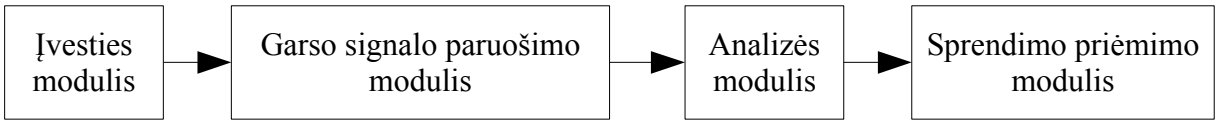


3.3.1 pav. Repozitorijų schema

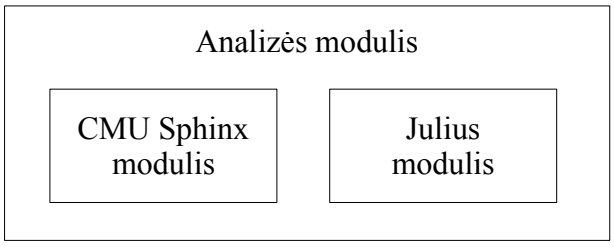
3.4. Sistemos architektūra

Kuriama atpažinimo sistema bus modulinės architektūros (3.4.1 pav.), ją sudarys tokios dalys:

- Įvesties modulis
- Garso signalo paruošimo modulis
- Analizės modulis (3.5.2.1 pav)
  - CMU Sphinx kalbos atpažinimo modulis
  - Julius kalbos atpažinimo modulis
- Sprendimo priėmimo modulis



3.4.1 pav. Sistemos architektūra



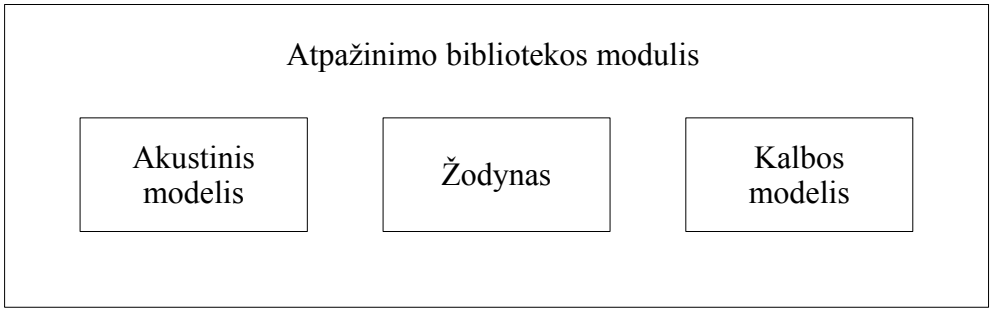
3.4.2 pav. Analizės modulis

Įvesties modulis paslepia vidinę garso įrenginio realizaciją ir suteikia patogią sąsają garso įrašo kūrimui.

Garso signalo paruošimo modulis paruošia signalą, nufiltruoja triukšmus ir tuščius nekalbinius intarpus.

Analizės modulis – susideda iš dviejų kalbos atpažinimo modulių (3.4.3 pav.): CMU Sphinx ir Julius. Analizės modulyje esančių kalbos atpažinimo modulių sąsajos yra tokios pačios ir jiems, kaip parametrai yra paduodami akustiniai ir kalbos modeliai, bei žodynas.

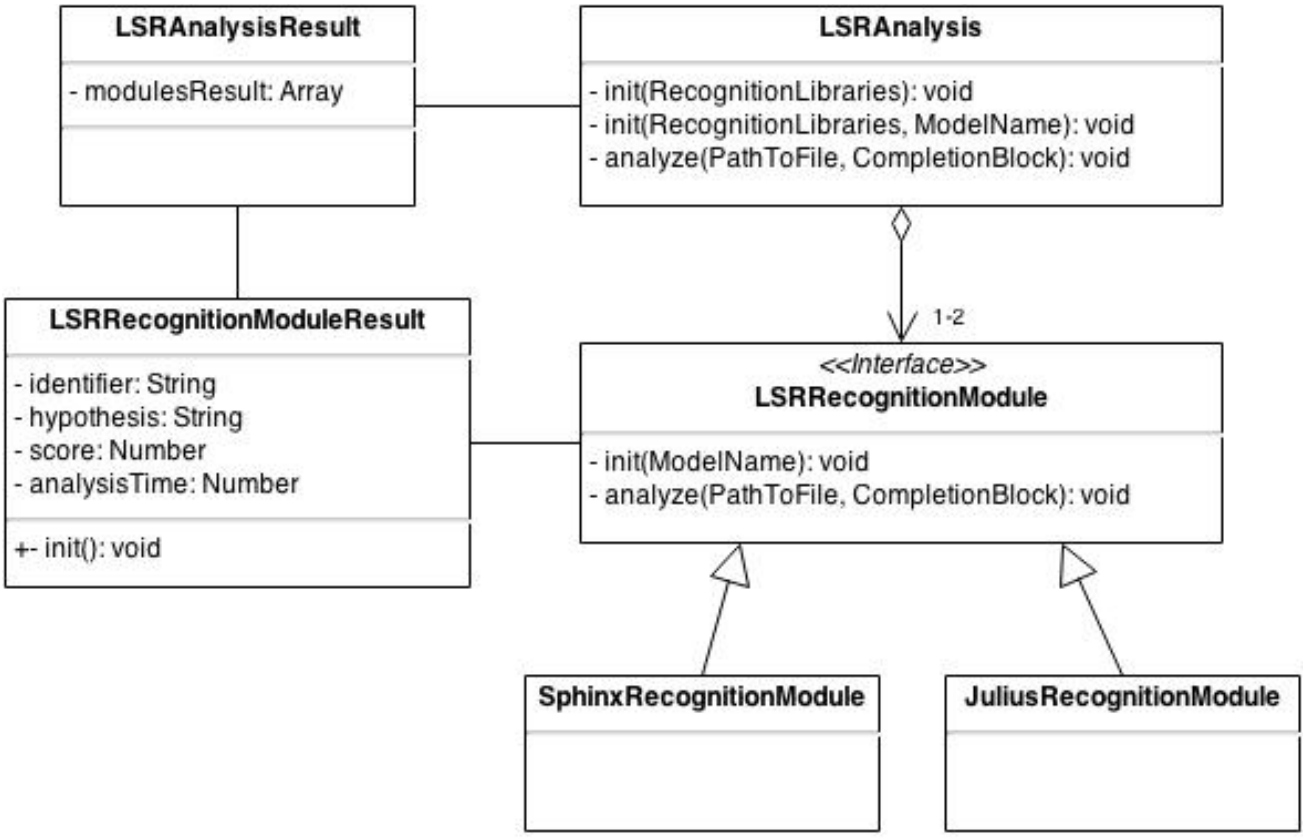
Sprendimo priėmimo modulis – priima sprendimą koks buvo ištartas žodis apibendrinus iš analizės gautus rezultatus.



3.4.3 pav. Atpažinimo bibliotekos modulis

Sistemą sudarys 6 pagrindinės klasės (3.4.4 pav.):

- LSRAnalysis – pagrindinė klasė per kurią vyksta įrašų analizė iš programėlės.
- LSRAnalysisResult – analizės rezultatų klasė.
- LSRRecognitionModule – unifikuotas interfeisas analizės moduliams.
- LSRRecognitionModuleResult – atpažinimo modulio rezultatų klasė.
- SphinxRecognitionModule – CMU Sphinx biblioteką naudojantis atpažinimo modulis.
- JuliusRecognitionModule – Julius biblioteką naudojantis atpažinimo modulis.



3.4.4 pav. UML klasių diagrama LSR karkasui

3.5. Sprendimo realizacija

Realizuojant CMU Sphinx bibliotekos modulį paaiškėjo, kad esami CMU Sphinx kompiliavimo skriptai į statinę iOS biblioteką nėra atnaujinti ir neveikia naujausiai iOS versijai. Pakoregavus šiuos skriptus pavyko sukompiliuoti CMU Sphinx į statinę biblioteką. Vėliau paaiškėjo, kad skriptai sukuria tik 32 bitų statinę biblioteką, kuri neveikia su 64 bitų architektūra (nuo A7 lustų). Todėl buvo nuspręsta nenaudoti šių skriptų ir kompiliuoti CMU Sphinx programinį kodą patiems. Patiems kompiliuojant CMU Sphinx sukuriama statinė biblioteka, kuri viduje turi atskiras 32 ir 64 bitų versijas. Be to buvo kompiliuojamas tik atpažinimui reikalingas programinis kodas.

Julius biblioteka nepalaiko iOS platformos, todėl Julius programinį kodą teko kompiliuoti į statinę biblioteką (su 32 ir 64 bitų palaikymu) patiems. Kompiliuojant susidurta su problema, kad kompiliatorius trūkstamus return šaukinius laiko klaidomis, todėl šios kodo vietos buvo pataisytos Julius programiniame kode (4 priedas). Be to buvo kompiliuojamas tik tas kodas, kuris nėra pririštas prie asmeninio kompiuterio operacinės sistemos (Windows, OS X), nes Julius yra skirtas asmeniniams kompiuteriams, o kompiliuojant nuo operacinės sistemos priklausantį kodą gaunamos klaidos, nes nerandamos importuojamos asmeninio kompiuterio operacinės sistemos klasės.

3.5.1. Reikalingų failų generavimas

Akustinių modelių, žodyno ir kalbos modelių kūrimui buvo sukurti Perl kalba parašyti skriptai, kurių pagalba automatiškai sukuriami reikalingi failai ir failų rinkiniai. Kad šie skriptai veiktų operacinėje sistemoje turi būti įdiegtos CMU Sphinx, Julius ir HTK bibliotekos ir žinoma Perl kalbos kompiliatorius. HTK yra naudojamas sukurti akustinius modelius Julius bibliotekai. Verta paminėti, kad HTK nepalaiko UTF-8 kodavimo, todėl lietuviškos raidės turi būti pakeičiamos ASCII simboliais pvz. š pakeičiama sh. Taip pat HTK naudojama naujausia versija 3.4.1 yra išleista 2009-05-13, kas reiškia, kad ši programinė įranga šiai dienai yra neatnaujinta virš 4 metų, kas kelia abejonių dėl tolesnio HTK palaikymo. Tuo tarpu, CMU Sphinx apmokymo modulis sphinxtrain palaiko UTF-8 kodavimą ir nereikia jokių papildomų konvertavimų iš UTF-8 į ASCII.

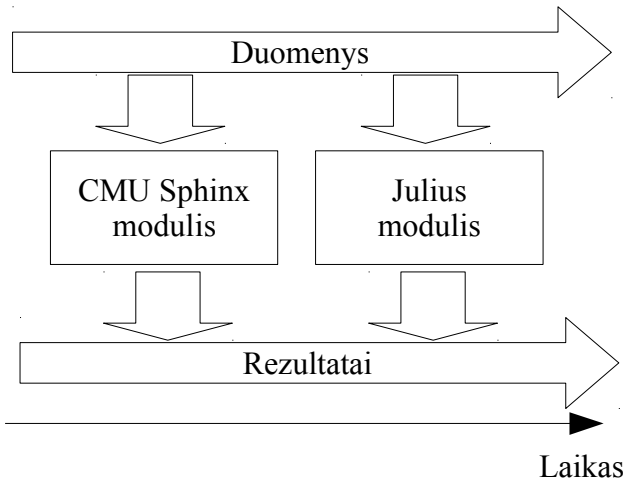
Paleidus automatizuotus apmokymo skriptus yra sugeneruojami akustinio modelio, žodyno, bei kalbos modelio failai, kuriuos įkėlus į kuriamą kalbos atpažinimo sistemą galima iškart naudoti atpažinimui.

3.5.2. Atpažinimo karkasas

Atpažinimui iOS įrenginiuose sukurtas LSR karkasas, kuris naudoja CMU Sphinx ir Julius bibliotekas. Taip pat šis karkasas sukompiliuojamas į statinę biblioteką, kurią galima naudoti kuriant atskiras programėles norinčias naudoti lietuvių kalbos atpažinimą.

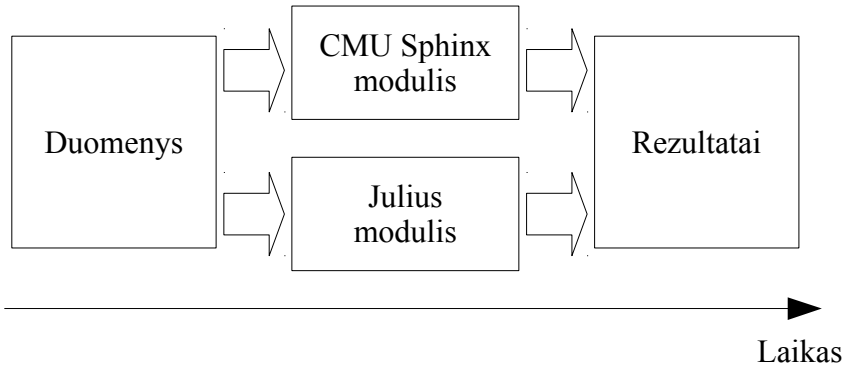
LSR karkase CMU Sphinx ir Julius bibliotekų realizacija bus paslėpta po atskirais moduliais turinčiais vienodą sąsają. Tai leidžia naudoti atpažinimo bibliotekas vienodai, nesigilinant į konkrečius kiekvienos bibliotekos niuansus. Taip pat ateityje bus galima prijungti ir daugiau atpažinimo bibliotekų į analizės modulį paslėpus realizaciją po bendra sąsaja.

Analizės modulis gali naudoti abu (CMU Sphinx ir Julius) atpažinimo modulius kartu. Tačiau dabartinėje realizacijoje atpažinimas naudojant abu modulius vykdomas nuosekliai (3.5.2.1 pav.), iš pradžių vienas modulis atpažįsta audio failą, po to kitas.



3.5.2.1 pav. Nuoseklus atpažinimas

Kadangi naujų iOS įrenginių procesoriai pradedant A6 turi daugiau, nei vieną branduolį yra galimybė atpažinimo bibliotekų darbą lygiagretinti, (3.5.2.2 pav.) tačiau reiktų atkreipti dėmesį į tai, kad šios bibliotekos parašytos su C ir jeigu bibliotekų realizacijose kokie nors veiksmai jau yra lygiagretinami gali atsirasti anomalijų ir veikimas būtų neprognozuojamas ir sistema lūžti. Nepaisant galimų trukdžių atpažinimo išlygiagretinimui atsiveria nemaža galimybė naudoti ne vieną atpažinimo biblioteką neprarandant atpažinimo greičio.

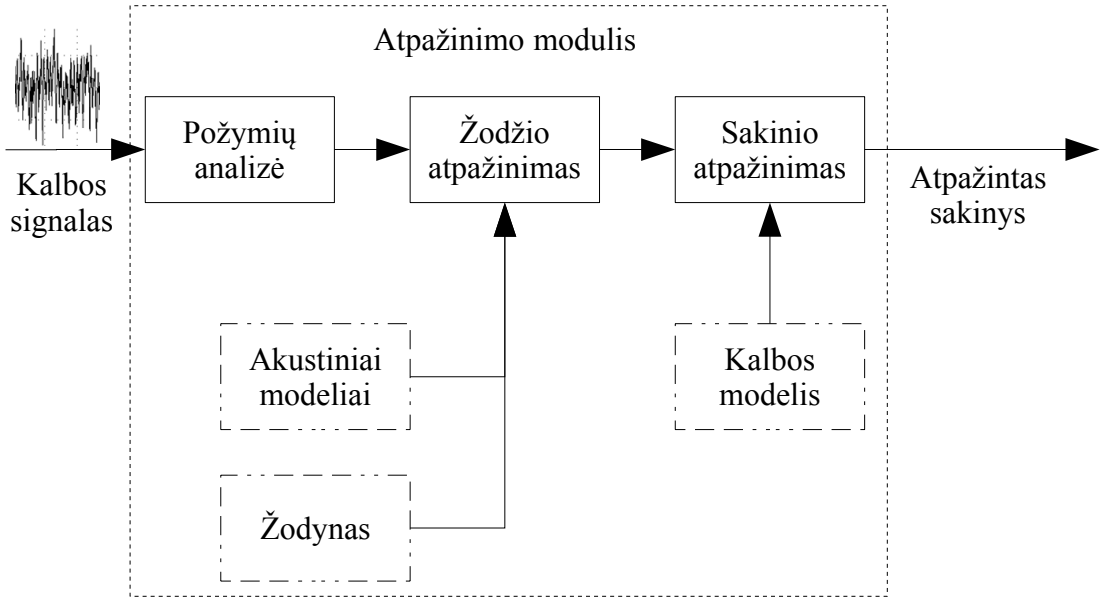


3.5.2.2 pav. Lygiagretus atpažinimas

3.5.3. Kalbos atpažinimas

Kalbos atpažinimas vyksta atpažinimo modulyje, jį atlieka CMU Sphinx arba Julius biblioteka. Atpažinimo veikimo principas pavaizduotas 3.5.3.1 pav. Kaip matome iš paveikslo pirmiausia iš kalbos signalo yra išskiriami požymiai, kurie PMM atitinka stebėjimus. Pagal šiuos požymius PMM nustato koks žodis buvo ištartas. Žodžio atpažinimui PMM naudoja akustinius modelius, kurie atitinka PMM būsenas, o žodynas naudojamas nustatyti konkrečius galimus perėjimus tarp būsenų. Kai yra turimi atpažinti žodžiai, pasinaudojus kalbos modeliu yra bandoma atpažinti koks tikėtinas sakinys buvo ištartas, pavyzdžiui sakinys „katinas skraido“ daug mažiau tikėtinas, nei „katinas rainas“.

Jei žodyne nebus žodžio, kuris buvo ištartas atpažinimo modulis rezultate pateiks žodį, kuris labiausiai akustiškai panašus į ištartą. Taip pat, jei kalbos modelyje nebus apibrėžtas ištartas sakinys atpažinimo modulis grąžins labiausiai panašų (tikėtiną) sakinį. Tai reiškia, kad atpažinimo modulis negali atpažinti dar nematytų (angl. o*ut of vocabulary*) žodžių.



3.5.3.1 pav. Atpažinimo schema

3.6. Apibendrinimas ir išvados

1. CMU Sphinx biblioteka daug geriau pritaikyta iOS platformai, nes turi skriptus, kurie paruošia CMU Sphinx biblioteką veikimui iOS platformoje, tačiau šie skriptai ganėtinai pasenę ir be korekcijos neveikia, tačiau palaiko tik 32 bitų architektūrą. Todėl karkaso realizacijoje nuspręsta CMU Sphinx biblioteką kompiliuoti iš kodo, nes norint palaikyti skirtingas iOS įrenginių procesorių architektūras reikia statinės bibliotekos, kuri viduje turėtų atskiras versijas 32 bitų ir 64 bitų architektūroms.
2. Julius biblioteką kompiliuojant į statinę biblioteką, kuri veiktų iOS platformoje neapsieita be pačios bibliotekos programinio kodo korekcijos, tačiau korekcijos buvo labai nežymios – naujausias kompiliatorius trūkstamus return šaukinius traktuodavo, kaip klaidas (4 priedas).
3. Tiek CMU Sphinx, tiek Julius bibliotekų ne visas programinis kodas buvo kompiliuojamas, o tik tos dalys, kurios reikalingos atpažinimo veikimui. CMU Sphinx buvo kompiliuojami tik atpažinimui reikalingi įrankiai, o Julius tik tos klasės, kurios neturi priklausomybių asmeninių kompiuterių operacinėms sistemoms (Windows, OS X).
4. Atpažinimo bibliotekos reikalingi trys failų rinkiniai: akustiniai modeliai, žodynas, kalbos modelis, tačiau šių failų formatai priklausomai nuo bibliotekos skiriasi. Julius naudoja HTK akustinių modelių ir žodyno formatą, bei ARPA standarto kalbos modelį. Tuo tarpu CMU Sphinx akustiniams modeliams naudoja savo formatą, žodynui paprastą formatą (eilutėje žodis tarpas fonetinė transkripcija), o kalbos modeliui JSGF (angl., *Java Speech Grammar Format*) formatą.
5. Akustinių modelių kūrimui Julius biblioteka neturi įrankių, todėl naudojamas HTK. Be to Julius nepalaiko UTF-8 kodavimo, kas sukuria papildomo konvertavimo darbo iš UTF-8 į ASCII. Šis konvertavimas atliekamas pakeičiant lietuviškus UTF-8 simbolius ASCII simboliais.
6. Git submoduliai naudojami atpažinimo bibliotekoms leidžia atskirti šias bibliotekas nuo LSR karkaso ir lengvai valdyti jų atnaujinimą, bei koregavimą izoliuojant bibliotekų programinį kodą nuo karkaso kodo ir leidžiant sekti pasikeitimus tarp bibliotekų versijų ir atliktų korekcijų.
7. Išlygiagretinus atpažinimo modulių darbą daugiaprocesorinėse sistemose būtų galima per tą patį laiką gauti kelis spėjimus ir pateikti galutinį rezultatą vartotojui apibendrinant abiejų atpažinimo bibliotekų rezultatus.

4. EKSPERIMENTINĖ DALIS

4.1. Tyrimo eigos aprašymas

4.1.1. Garsynas

Tyrimai bus atliekami naudojantis daugelio kalbėtojų garsynu, kuris priklauso KTU kalbos signalų tyrimo laboratorijai, jo autoriai K. Ratkevičius, V. Rudžionis ir R. Maskeliūnas. Garsyną sudaro 50 vyrų ir 50 moterų nuo 16 iki 18 metų ištartų skaičių įrašai 16 bitų WAV formatu. Iš viso 10349 įrašai, kurių bendra trukmė 2 valandos 45 minutės 28 sekundės.

4.1.2. Automatinis kalbos atpažinimo tikslumo vertinimas

Darbe bus tiriamas pavienių skaičių atpažinimo tikslumas, kur žodžio klaidos procentas – WER (angl., *Word Error Rate*) bus skaičiuojamas pagal

$$WER = \left(1 - \frac{n_{teisingi}}{n}\right) * 100 \quad (47)$$

Kur $n_{teisingi}$ yra teisingų hipotezių skaičius, o n – eksperimentų skaičius.

4.1.3. Įrenginiai

Tyrimas bus atliekamas su kiek įmanoma daugiau skirtingų iOS įrenginių ir vienu asmeniniu kompiuteriu (4.1.3.1 lentelė), nes procesoriaus galingumas (skaičiavimų pajėgumas) turi didelę įtaką, kaip greitai galima atpažinti signalą. Tas pats kalbos signalas ant poros metų senumo įrenginio su žymiai lėtesniu procesoriumi bus atpažįstamas, kur kas lėčiau, nei ant kelių mėnesių senumo įrenginiu su pačiu naujausiu procesoriumi. Be to Apple orientuojasi į aukštos klasės (angl., *high-end*) įrenginius ir kiekvienais metais stengiasi išleisti vis galingesnius įrenginius, taigi tikėtina, kad su pačiais naujausiais įrenginiais bus pasiektas kur kas geresnis atpažinimo greitis, nei su senesniais.

4.1.3.1 lentelė. Įrenginių sąrašas

Įrenginys	**Procesorius**	**Atmintis**	**Lustas**
iPod Touch 5th	1 GHz (underclocked to 800 MHz) dual-core ARM Cortex-A9	512 MB	Apple A5
iPhone 4S	1 GHz (underclocked to 800 MHz) dual-core ARM Cortex-A9	512 MB LPDDR2 DRAM	Apple A5
iPhone 5/iPhone 5C	1.3 GHz dual-core Apple-designed ARMv7s	1 GB LPDDR2 DRAM	Apple A6
iPhone 5S	1.3 GHz dual-core Apple-designed ARMv8-A 64-bit	1 GB LPDDR3 DRAM	Apple A7
iPad 2	1 GHz dual-core ARM Cortex-A9	512 MB DDR2 RAM (1066 Mbit/s data rate) built into Apple A5 package	Apple A5
iPad 3th	1 GHz dual-core ARM Cortex-A9	1024 MB LPDDR2 RAM	Apple A5X
iPad Air	1.4 GHz dual-core Apple Cyclone	1024 MB LPDDR3 RAM	Apple A7
iPad Mini	1 GHz dual-core ARM Cortex-A9	512 MB DDR2 RAM built into Apple A5 package	Apple A5
Mac mini (OS X operating system)	2.3GHz quad-core Intel Core i7 (Turbo Boost up to 3.3GHz) with 6MB L3 cache	2x8 Gb DDR3 RAM (1600 MHz)	

4.1.4. Eksperimento eiga

Tyrimui sukurti 9 trifonų akustiniai modeliai skaičiams nuo 0 iki 9 su skirtingų kalbėtojų skaičiumi ir įrašų vienam kalbėtojui skaičiumi:

- 50x16 – 50 kalbėtojų po 16 įrašų vienam kalbėtojui.
- 50x4 – 50 kalbėtojų po 4 įrašus vienam kalbėtojui.
- 50x1 – 50 kalbėtojų po 1 įrašą vienam kalbėtojui.
- 30x16 – 30 kalbėtojų po 16 įrašų vienam kalbėtojui.
- 30x4 – 30 kalbėtojų po 4 įrašus vienam kalbėtojui.
- 30x1 – 30 kalbėtojų po 1 įrašą vienam kalbėtojui.
- 10x16 – 10 kalbėtojų po 16 įrašų vienam kalbėtojui.
- 10x4 – 10 kalbėtojų po 4 įrašus vienam kalbėtojui.
- 10x1 – 10 kalbėtojų po 1 įrašą vienam kalbėtojui.

CMU Sphinx ir Julius bibliotekai šie 9 akustiniai modeliai buvo kuriami skirtingai. CMU Sphinx bibliotekai akustiniai modeliai buvo kuriami naudojant CMU Sphinx įrankius pagal aprašymą iš CMU Sphinx tinklapio [32], CMU Sphinx akustinių modelių kūrimo konfigūracinis failas pateiktas 1 priede. Julius bibliotekai akustiniai modeliai buvo kuriami naudojantis HTK pagal aprašymą iš VoxForge tinklapio [33], HTK akustinių modelių kūrimo konfigūracinis failas pateiktas 2 priede.

Virš LSR karkaso buvo sukurta testinė programėlė, kuri naudodama šį karkasą (jo viduje CMU Sphinx ir Julius bibliotekas) analizavo testinių įrašų rinkinį, kuris susidarė iš 50 kalbėtojų įrašų po 5 įrašus kalbėtojui (iš viso 2500 įrašų). Kiekvienas akustinis modelis buvo testuojamas su šiuo testinių įrašų rinkiniu.

LSR karkaso rezultatai kiekvienam akustiniam modeliui buvo saugomi atskiruose CSV formato failuose, po vieną failą CMU Sphinx ir Julius bibliotekai. Kiekvieną CSV failą sudarė 2500 eilučių, kur kiekvienoje buvo kableliais atskirtos reikšmės (failo pavyzdys 3 priede):

- bibliotekos pavadinimas (Sphinx arba Julius),
- failo pavadinimas,
- failo trukmė sekundėmis,
- atpažinimo laikas sekundėmis,
- failo transkripcija,
- analizės hipotezė,
- analizės teisingumas (0, jei hipotezė neteisinga ir 1, jei teisinga)

Šie analizės rezultatų CSV failai buvo importuoti į LibreOffice Skaičiuoklę ir analizuoti, šių failų analizės rezultatai pateikti tolesniuose skyriuose.

4.2. Apmokymo duomenų dydžio ir akustinio modelio įtaka atpažinimo tikslumui

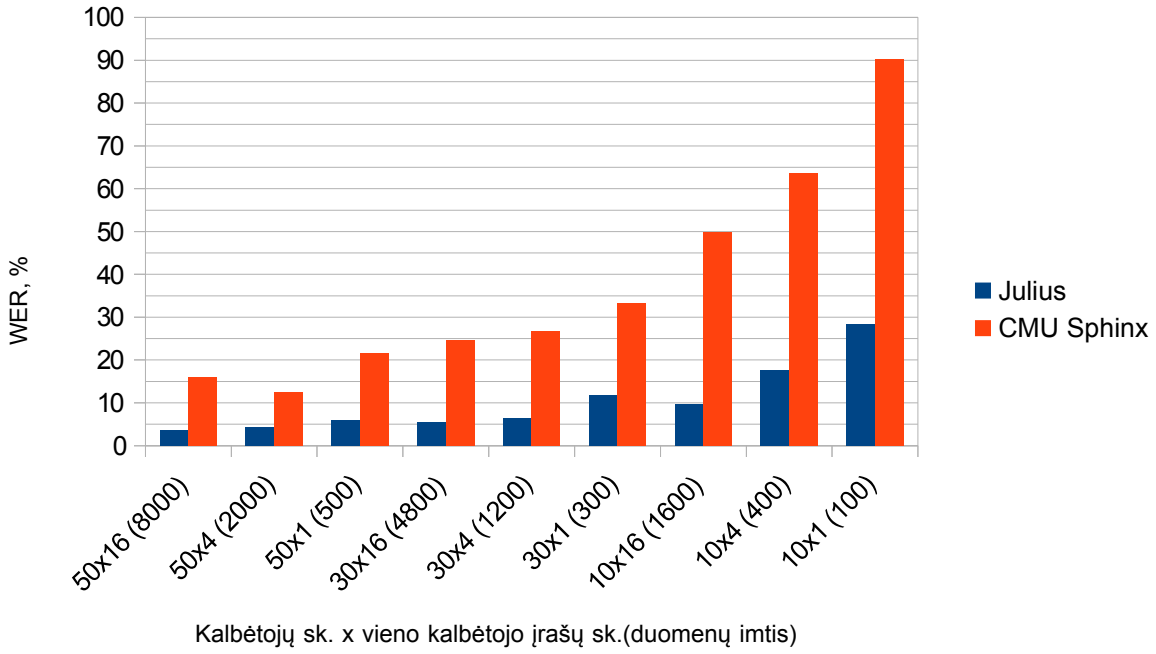
Norint išsiaiškinti kaip žodžio klaidos procentas – WER (angl., *Word Error Rate*) priklauso nuo apmokymo duomenų imties ir akustinio modelio buvo atlikti eksperimentai. Eksperimento metu buvo tiriami visi 9 akustiniai modeliai, kurių duomenų imtis buvo skirtinga priklausomai nuo kalbėtojų ir failų vienam kalbėtojui skaičiaus. Gauti rezultatai pateikti 4.2.1 lentelėje.

4.2.1 lentelė. WER skirtingiems akustiniams modeliams ir duomenų imtims

Akustinis modelis	Duomenų imtis	Julius WER	CMU Sphinx WER
50x16	8000	3.53	15.83
50x4	2000	4.33	12.42
50x1	500	5.82	21.64
30x16	4800	5.41	24.69
30x4	1200	6.45	26.65
30x1	300	11.70	33.35
10x16	1600	9.70	49.94
10x4	400	17.68	63.53
10x1	100	28.42	90.30

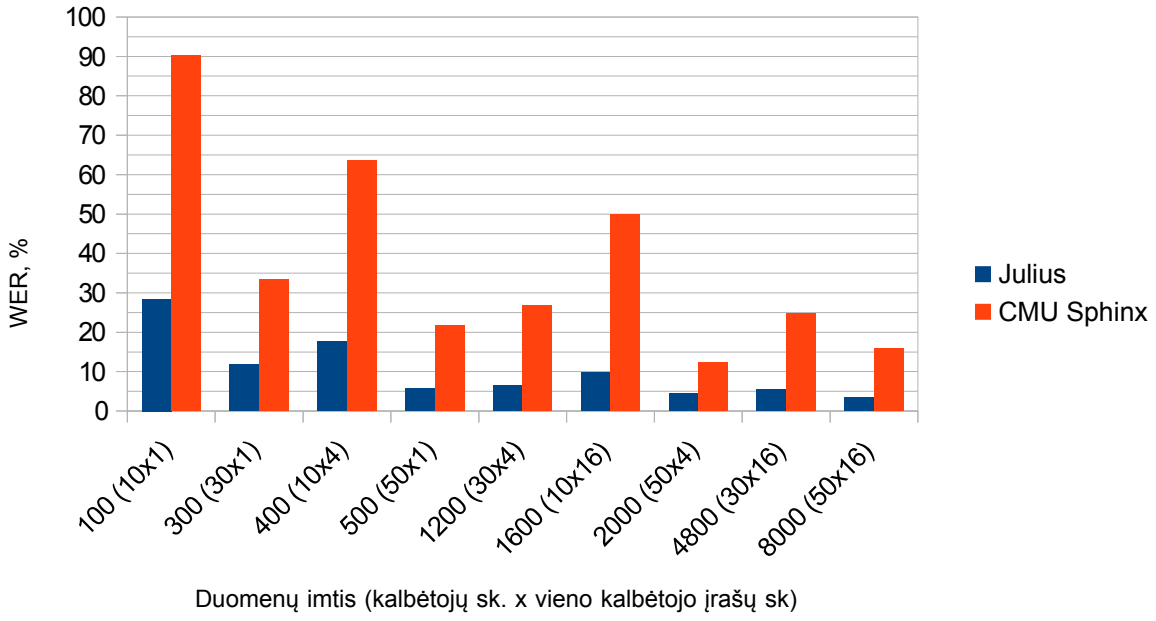
Iš rezultatų matyti, kad Julius biblioteka vidutiniškai 3.18 karto geriau atpažino testinius skaičius, nei CMU Sphinx biblioteka. Šie rezultatai sako, kad akustinių modelių kūrimas Julius bibliotekai naudojant HTK įrankį yra žymiai geriau aprašytas ir be papildomų pastangų galima pasiekti realiems taikymams priimtiną tikslumą, kai tuo tarpu CMU Sphinx bibliotekos akustinių modelių kūrimas pagal aprašymą duoda pakankamai prastus rezultatus ir reikia įdėti papildomų pastangų perskaičiuojant (angl., *reestimating*) akustinius modelius, kad būtų pasiektas geras atpažinimo tikslumas. Be to ši akustinių modelių perskaičiavimo procedūra CMU Sphinx bibliotekai nėra pažingsniui aprašyta, kaip tai yra HTK įrankiui.

Taip pat iš rezultatų matyti, kad tiek Julius tiek CMU Sphinx atpažinimo tikslumas daug labiau priklauso kalbėtojų skaičiaus, nei nuo duomenų imties ir kuo daugiau kalbėtojų, tuo tai labiau tiesa (4.2.1 pav.). Tai reiškia, kad kiekvienas naujas kalbėtojas duoda daug daugiau naudos sistemos tikslumui, nei tų pačių žodžių ištartų to pačio kalbėtojo skaičius. Pavyzdžiui Julius 50x1 akustinio modelio duomenų imtis yra vos 500 įrašų ir WER 5.8% (Julius modeliams), kai tuo tarpu 50x16 akustinio modelio duomenų imtis 8000 įrašų ir WER 3.5% (Julius modeliams). Tai reiškia, kad 50x16 modelio apmokymo duomenų imtis 16 kartų didesnė, nei 50x1, o tikslumas pagerėjo vos 2.3%. Taigi sudarant apmokymo duomenų rinkinį pravartu didinti kalbėtojų skaičių, o ne tų pačių žodžių įrašų kalbėtojui skaičių. Naudojantis šią taktika galima ženkliai sumažinti apmokymo trukmę nežymiai prarandant atpažinimo tikslumo, nes testinių įrašų skaičius tiesiogiai proporcingas apmokymo trukmei.



4.2.1 pav. Atpažinimo bibliotekų WER pagal akustinius modelius

Nepaisant to, kad kalbėtojų skaičiaus didėjimas ženkliau mažina WER, nei tų pačių vieno kalbėtojo žodžių įrašų skaičius, įrašų kiekio didėjimas turi tendenciją mažinti WER (4.2.2 pav.). Tačiau ši tendencija labiau ryški Julius akustiniams modeliams. Pavyzdžiui 30x16 akustinio modelio, kurio apmokymo duomenų imtis 4800, WER yra 5.4 % (Julius modeliams), kas nežymiai (vos vienu procentu) nusileidžia 50x4 akustinio modelio su 2000 apmokymo duomenų imtimi WER, kuri yra 4.3% (Julius modeliams).



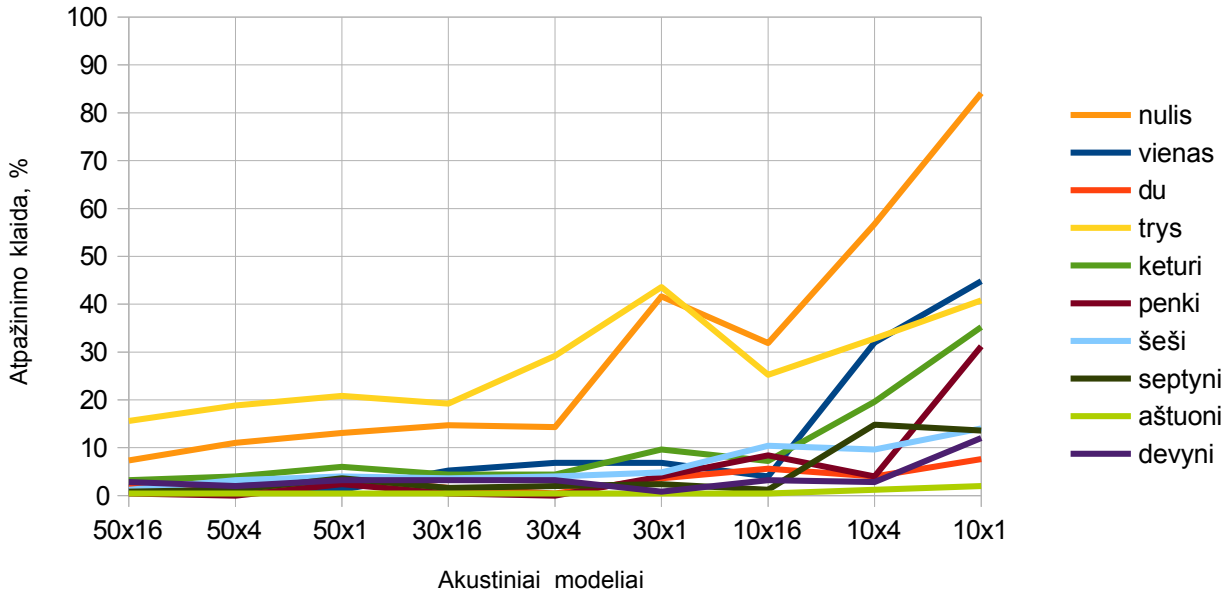
4.2.2 pav. Atpažinimo bibliotekų WER pagal duomenų imtį

4.3. Atskirų skaičių atpažinimo tikslumas

Patikrinus, kaip bibliotekos atpažįsta atskirus skaičius paaiškėjo, jog Julius akustiniai modeliai gan gerai atpažįsta daugelį skaičių testiniuose įrašuose išskyrus nulį ir tris (4.3.1 lentelė ir 4.3.1 pav.). Tai gali reikšti, kad apmokymo duomenyse nebuvo pakankama įvairovė šių skaičių ištarimų, kad sukurtas modelis pakankamai gerai apibendrintų šiuos skaičius.

4.3.1 lentelė. Julius skaičių atpažinimo klaida

	Vienas	**Du**	**Trys**	**Keturi**	**Penki**	**Šeši**	**Septyni**	**Aštuoni**	**Devyni**	**Nulis**
50x16	1.6	2.0	15.6	3.2	0.4	1.2	0.8	0.4	2.8	7.3
50x4	2.4	0.4	18.8	4.0	0.0	3.2	1.2	0.4	2.0	11.0
50x1	1.2	3.6	20.8	6.0	2.4	4.0	3.6	0.4	3.2	13.1
30x16	5.2	1.6	19.2	4.4	0.4	3.6	1.6	0.4	3.2	14.7
30x4	6.8	0.4	29.2	4.4	0.0	4.0	2.0	0.4	3.2	14.3
30x1	6.8	3.6	43.6	9.6	4.0	4.8	2.4	0.4	0.8	41.6
10x16	4.0	5.6	25.2	7.2	8.4	10.4	1.2	0.4	3.2	31.8
10x4	32.0	4.0	32.8	19.6	4.0	9.6	14.8	1.2	2.8	56.7
10x1	44.8	7.6	40.8	35.2	31.2	14.0	13.6	2.0	12.0	84.1

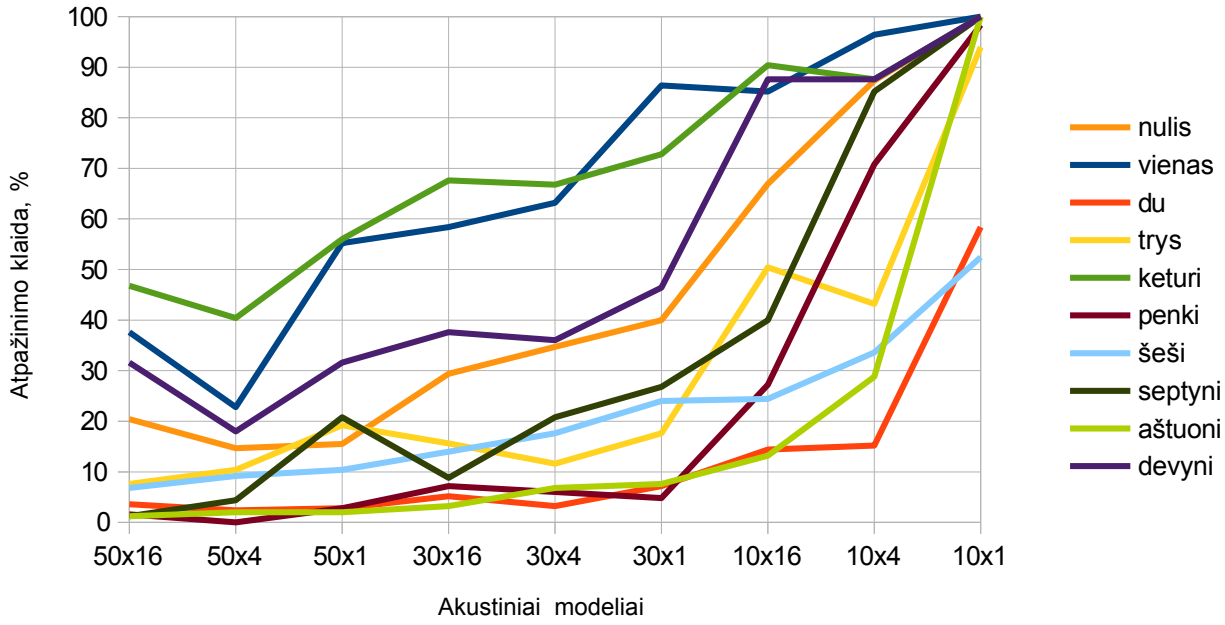


4.3.1 pav. Julius skaičių atpažinimo klaida

Kaip matome iš 4.3.2 lentelės ir 4.3.2 pav., kad CMU Sphinx akustiniai modeliai prasčiausiai atpažįsta skaičius vienas ir keturi. CMU Sphinx atskirų skaičių atpažinimo klaidos pakankamai įvairios ir sunku vertinti, kas galėjo tai įtakoti, kai tuo tarpu Julius atskirų skaičių atpažinime ryškiai išsiskiria nulis ir trys.

4.3.2 lentelė. CMU Sphinx skaičių atpažinimo klaida

	Vienas	Du	Trys	Keturi	Penki	Šeši	Septyni	Aštuoni	Devyni	Nulis
50x16	37.6	3.6	7.6	46.8	1.6	6.8	1.2	1.2	31.6	20.4
50x4	22.8	2.4	10.4	40.4	0.0	9.2	4.4	2.0	18.0	14.7
50x1	55.2	2.8	19.2	56.0	2.8	10.4	20.8	2.0	31.6	15.5
30x16	58.4	5.2	15.6	67.6	7.2	14.0	8.8	3.2	37.6	29.4
30x4	63.2	3.2	11.6	66.8	6.0	17.6	20.8	6.8	36.0	34.7
30x1	86.4	7.2	17.6	72.8	4.8	24.0	26.8	7.6	46.4	40.0
10x16	85.2	14.4	50.4	90.4	27.2	24.4	40.0	13.2	87.6	66.9
10x4	96.4	15.2	43.2	87.6	70.8	33.6	85.2	28.8	87.6	87.3
10x1	100.0	58.4	94.0	100.0	98.4	52.4	100.0	100.0	100.0	100.0



4.3.2 pav. CMU Sphinx skaičių atpažinimo klaida

4.4. Įrenginio įtaka atpažinimo greičiui

Šiame skyriuje pristatomi eksperimentai, kuriais siekta įvertinti skirtingų įrenginių įtaką atpažinimo greičiui. Visi įrenginiai buvo testuojami su skirtingais akustiniais modeliais pradedant 10x1 ir baigiant 50x16. Eksperimento rezultatai pateikti 4.4.1 ir 4.4.2 lentelėse. Lentelėse pateikti vidutiniai atpažinimo laikai vienai įrašo sekundei.

$$Vidutinis\ atpažinimo\ laikas = \frac{\sum_{i=1}^{n}\left(\frac{Atpažinimo\ laikas}{Įrašo\ ilgis}\right)}{n} \tag{48}$$

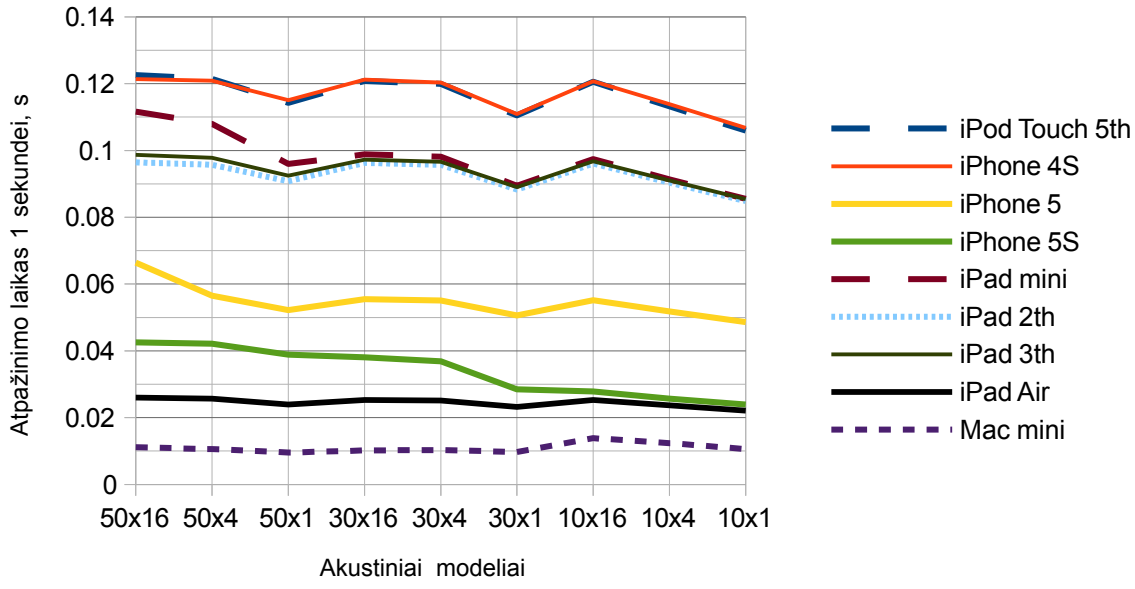
Kaip matome iš rezultatų kuo naujesnis ir galingesnis įrenginio procesorius tuo greičiau buvo atpažinti testiniai įrašai. Geriausi rezultatai buvo pasiekti su iPhone 5, iPhone 5S ir iPad Air įrenginiais (4.4.1 pav. ir 4.4.2 pav.). Verta paminėti, kad iPhone 5S ir iPad Air įrenginiai yra ne 32, o 64 bitų architektūros ir yra šiai dienai naujausi Apple įrenginiai savo kategorijoje (telefonų ir

planšetinių kompiuterių). Be to iPod Touch 5th nors ir yra naujausias Apple įrenginys daugiafunkcinio kišeninio kompiuterio kategorijoje, tačiau jo greitaveika atitinka iPhone 4S, nes jų procesoriai yra identiški. Taip pat iPad mini, iPad 2th ir iPad 3th greitaveikos labai panašios, nes jų procesoriai identiški išskyrus iPad 3th, kurio procesorius truputį modifikuotas, bet taip pat tos pačios kartos. Dar matome, kad iPod Touch 5th ir iPhone 4S greitaveika yra mažesnė, nei iPad mini, iPad 2th ir iPad 3th, ir nors jų lustai yra tos pačios kartos, tačiau iPod Touch 5th ir iPhone 4S įrenginių procesorių versijos yra sumažintų pajėgumų. Tą patį matome tarp iPhone 5S ir iPad Air, tai leidžia daryti išvadą, kad tarp iPhone ir iPad įrenginių su tos pačios kartos lustai visada yra greitaveikos skirtumas ir iPad įrenginiai visais atvejais veiks greičiau, nes jų procesoriai galingesni.

Lyginant iOS įrenginių greitaveiką su asmeniniu Mac mini kompiuteriu matome, kad asmeninis kompiuteris sugebėjo atpažinti testinius įrašus dvigubai greičiau, nei greičiausias iOS įrenginys – iPad Air. Julius bibliotekai Mac mini vidutiniškai buvo greitesnis 2.2 karto, o CMU Sphinx bibliotekai vidutiniškai 2.7 karto, lyginant su iPad Air. Tačiau iPad Air vidutinis atpažinimo laikas 1 sekundei įrašo nuo Mac mini skyrėsi 14 ms Julius bibliotekai ir 29 ms CMU Sphinx bibliotekai. Toks nedidelis laiko skirtumas trumpiems kelių sekundžių įrašams nesudaro pastebimo skirtumo atpažinimo sistemos vartotojui.

4.4.1 lentelė. Julius bibliotekos vidutinis atpažinimo laikas 1 sekundei įrašo

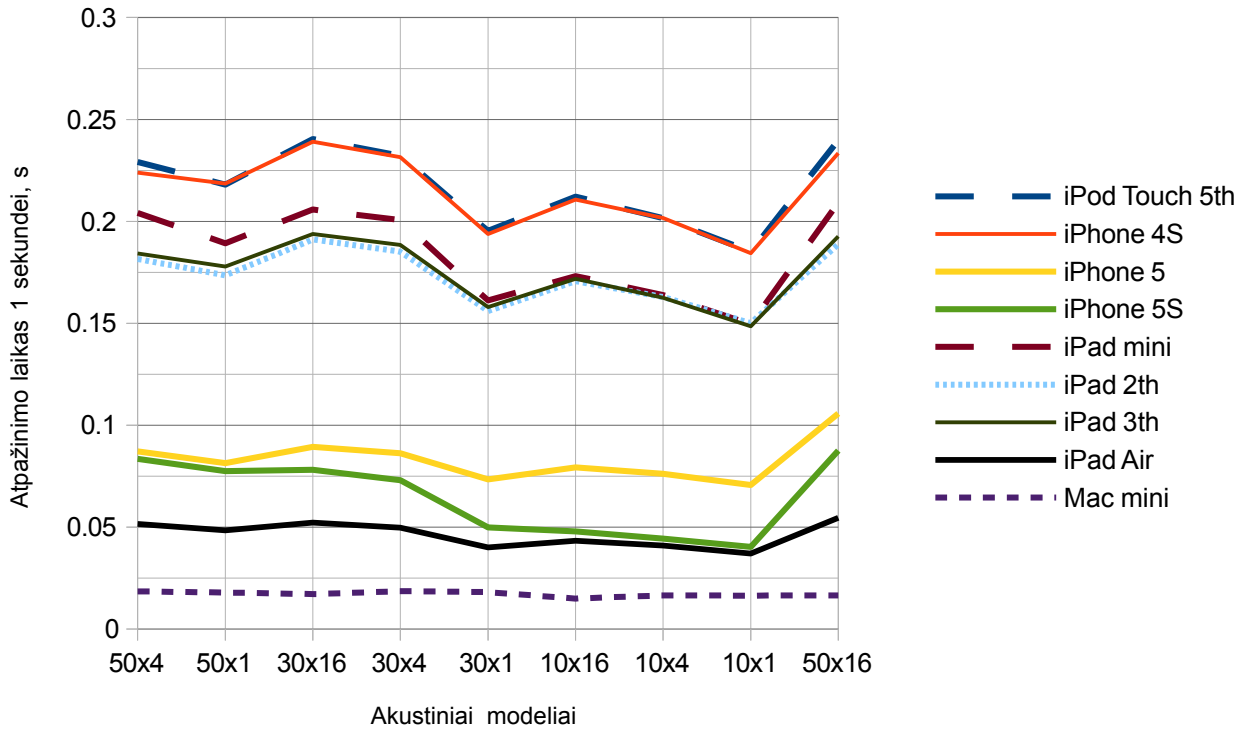
	50x16	**50x4**	**50x1**	**30x16**	**30x4**	**30x1**	**10x16**	**10x4**	**10x1**
iPod Touch 5th	0.123	0.121	0.114	0.121	0.120	0.111	0.121	0.113	0.106
iPhone 4S	0.121	0.121	0.115	0.121	0.120	0.111	0.121	0.114	0.107
iPhone 5	0.066	0.057	0.052	0.055	0.055	0.051	0.055	0.052	0.049
iPhone 5S	0.043	0.042	0.039	0.038	0.037	0.029	0.028	0.026	0.024
iPad mini	0.112	0.108	0.096	0.099	0.098	0.089	0.097	0.091	0.086
iPad 2th	0.096	0.096	0.091	0.096	0.096	0.088	0.096	0.090	0.085
iPad 3th	0.099	0.098	0.093	0.097	0.097	0.089	0.097	0.091	0.086
iPad Air	0.026	0.026	0.024	0.025	0.025	0.023	0.025	0.024	0.022
Mac mini	0.011	0.011	0.010	0.010	0.010	0.010	0.014	0.012	0.011



4.4.1 pav. Julius bibliotekos atpažinimo greitis

4.4.2 lentelė. CMU Sphinx bibliotekos vidutinis atpažinimo laikas 1 sekundei įrašo

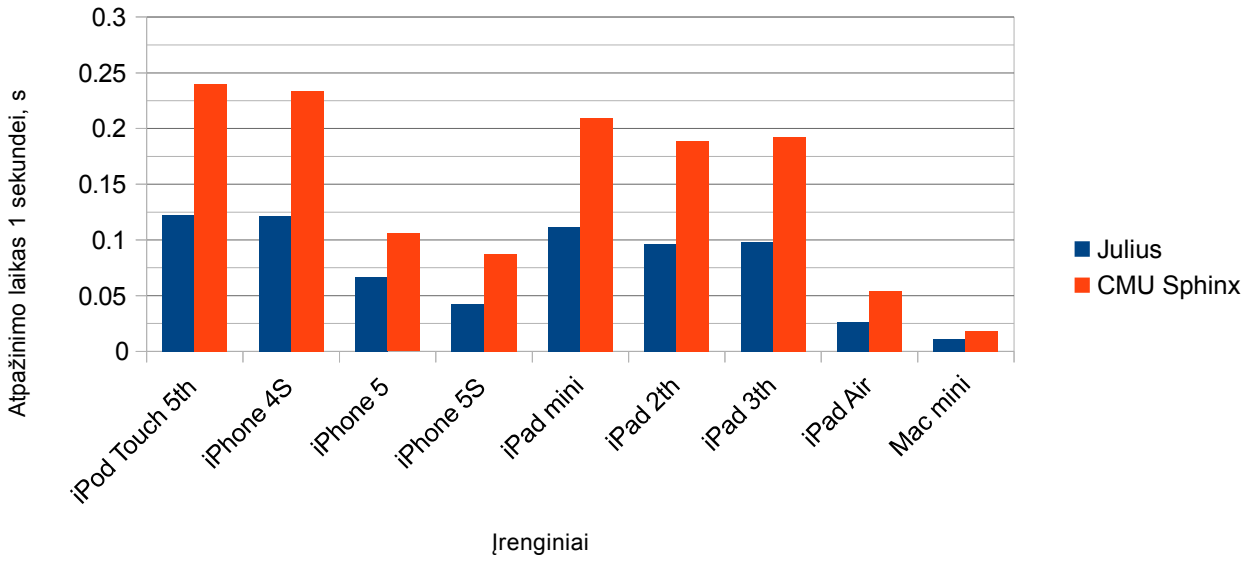
	50x16	50x4	50x1	30x16	30x4	30x1	10x16	10x4	10x1
iPod Touch 5th	0.240	0.229	0.218	0.240	0.232	0.195	0.212	0.202	0.185
iPhone 4S	0.233	0.224	0.219	0.239	0.232	0.194	0.211	0.202	0.184
iPhone 5	0.106	0.087	0.082	0.089	0.086	0.074	0.079	0.076	0.071
iPhone 5S	0.088	0.084	0.078	0.078	0.073	0.050	0.048	0.044	0.040
iPad mini	0.209	0.204	0.189	0.206	0.201	0.161	0.173	0.164	0.150
iPad 2th	0.189	0.182	0.174	0.191	0.185	0.156	0.171	0.163	0.150
iPad 3th	0.193	0.184	0.178	0.194	0.189	0.158	0.172	0.163	0.149
iPad Air	0.055	0.052	0.048	0.052	0.050	0.040	0.043	0.041	0.037
Mac mini	0.019	0.018	0.017	0.019	0.018	0.015	0.017	0.016	0.017



4.4.2 pav. CMU Sphinx bibliotekos atpažinimo greitis

Lyginant CMU Sphinx ir Julius bibliotekų atpažinimo greitaveiką (4.4.3 pav.) matome, kad Julius bibliotekos vidutinis atpažinimo laikas vienai sekundei įrašo yra beveik dvigubai geresnis, nei CMU Sphinx bibliotekos. Tačiau šie rezultatai gauti naudojant standartines bibliotekų konfigūracijų reikšmes, kas gali reikšti, kad Julius bibliotekos standartinės reikšmės yra labiau orientuotos į greitaveiką, nei CMU Sphinx, taip pat Julius programinis kodas gali būti labiau optimizuotas, skirtis naudojamų metodų realizacijos ar pačių metodų variacijos. Todėl vienareikšmiškai teigti, kad Julius biblioteka greitesnė, nei CMU Sphinx negalima, nes didelę įtaką atpažinimo greičiui turi bibliotekų realizacijos. Tačiau, galima teigti, kad su standartine konfigūracija Julius bibliotekos vidutinis

atpažinimo laikas įrašo sekundei yra beveik dvigubai geresnis, nei CMU Sphinx bibliotekos su standartine konfigūracija.



4.4.3 pav. CMU Sphinx ir Julius bibliotekų atpažinimo greičių palyginimas 50x16 akustiniam modeliui

4.5. Eksperimentų išvados

1. Julius akustiniai modeliai sukurti su HTK 3.18 karto tiksliau atpažįsta naujus dar nematytus įrašus, nei CMU Sphinx akustiniai modeliai. CMU Sphinx reikia papildomų pastangų perskaičiuojant akustinius modelius, kad būtų gauti panašūs rezultatai, tačiau ši perskaičiavimo procedūra nėra pažingsniui aprašyta, taip kaip HTK įrankiui.
2. Atpažinimo tikslumas tiesiogiai proporcingas kalbėtojų skaičiui ir tų pačių kalbėtojų pakartotinai ištarti tie patys skaičiai turi labai mažą naudą atpažinimo tikslumui, nes 50x1 akustinio modelio su 500 apmokymo įrašų WER lyginant su 50x4 (2000 apmokymo įrašų, 1500 papildomų ištarimų) akustiniu modeliu skiriasi viso labo 1.5%, o nuo 50x16 (8000 apmokymo įrašų, 7500 papildomų ištarimų) akustinio modelio 2.5%. Kai tuo tarpu 30x1 (300 apmokymo įrašų) nuo 50x1 (500 apmokymo įrašų) skiria 200 apmokymo įrašų, o WER pokytis 5.9%.
3. Atpažįstant atskirus skaičius su Julius labai išsiskiria skaičiai nulis ir trys, kurių atpažinimo klaidos didžiausios. Šias klaidas galėjo įtakoti nepakankama šių skaičių ištarimo įvairovė apmokymo įrašuose. Tuo tarpu CMU Sphinx biblioteka prasčiausiai atpažino skaičius vienas ir keturi, o skaičių atpažinimo klaidos pakankamai išsibarsčiusios, todėl sunku vertinti, kas galėjo įtakoti skaičių vienas ir keturi prastą atpažinimą.
4. Kiekviena nauja įrenginių su spartesniais procesoriais karta paspartina atpažinimo greitį, kartais net beveik du kartus ir ateityje galima tikėtis, kad iOS įrenginių atpažinimo greitis bus vis labiau artimas asmeniniam kompiuteriui, juolab, kad jau dabar trumpiems kelių sekundžių įrašams atpažinti ar su iPad Air ar su Mac mini sistemos vartotojas neturėtų pastebėti greitaveikos skirtumų.
5. Lyginant atpažinimo greitaveiką Julius pasirodė geriau, nei CMU Sphinx, bet tai gali būti susiję su standartinėmis šių bibliotekų konfigūracijomis, Julius programinio kodo optimizavimu ar implementacijoje naudojamų metodų variacijomis. Kadangi abiem bibliotekoms nebuvo naudoti identiški akustiniai modeliai ir bibliotekos veikė su standartinėmis konfigūracijomis negalima vienareikšmiškai vertinti šių bibliotekų greičių skirtumų.

5. DARBO REZULTATAI IR IŠVADOS

1. Julius biblioteka yra tinkama lietuvių kalbos atpažinimui, kai žinios šioje srityje nėra didelės, nes sukurti akustiniai modeliai su HTK (pagal standartinį aprašymą) jau iš kart duoda mažą WER (geriausiu atveju 3.52%). Be to pati Julius biblioteka su standartine konfigūracija veikia greičiau nei CMU Sphinx. Visgi didelis minusas yra tai, kad akustinių modelių kūrimas nėra integruotas į Julius ir reikia papildomai naudoti HTK. Dar vienas didelis minusas yra tas, kad sukurti akustiniai modeliai yra ASCII, o ne UTF-8 koduotės, kas reiškia, kad reikia įsivesti papildomą konvertavimo sistemą lietuviškiems simboliams.
2. CMU Sphinx iš kart turi akustinių modelių apmokymo įrankius ir nereikia naudoti dar vienos trečios šalies bibliotekos. Tačiau pagal standartinius aprašymus sukurti akustiniai modeliai atpažinimo tikslumu net 3.18 karto atsilieka nuo analogiškų Julius akustinių modelių kurtų su HTK, kas reiškia, kad reikia turėti žinių, kaip optimizuoti akustinių modelių kūrimą, nes gautas WER (geriausiu atveju 12.42%) nėra tinkamas realiems taikymams. Tačiau CMU Sphinx palaiko UTF-8 koduotę ir nereikia kurti papildomų konvertavimo mechanizmų.
3. Tiek Julius tiek CMU Sphinx bibliotekų atpažinimo greitaveika yra tinkama realiems mažo žodyno (10 komandų) pavienių žodžių atpažinimo taikymams, o gauti rezultatai (greičiausiai 1 sekundė įrašo atpažinta per 26ms) leidžia daryti prielaidą, kad ir didesniam nei 10 komandų žodynui greitaveika bus priimtina. Visgi išlieka klausimas, kur yra žodyno dydžio „lubos“.
4. Lyginant Julius ir CMU Sphinx bibliotekas, CMU Sphinx yra labiau patraukli lietuvių kalbos atpažinimui, nes viename pakete gaunamas žodžių atpažinimo ir apmokymo įrankis palaikantis UTF-8 koduotę. Šį CMU Sphinx paketą kadangi jis yra vientisas daug lengviau perkelti ant iOS platformos, nei du atskirus Julius ir HTK paketus. Be to CMU Sphinx jau dabar dalinai palaiko iOS platformą (palaikomas atpažinimas, o apmokymas dar ne), kai tuo tarpu Julius ir HTK yra skirti asmeniniams kompiuteriams. Apmokymo mechanizmo perkėlimas ant iOS yra svarbus ateities žingsnis norint pasiekti geresnių atpažinimo rezultatų, nes turint apmokymo modulį iOS platformoje galima adaptuoti turimus akustinius modelius konkrečiam kalbėtojui. Tai kad spartėjant įrenginiams vis daugiau skaičiavimų bus perkeliama į klientinę (įrenginio) dalį leis atsisakyti interneto prieigos, juolab, kad esant lėtam internetui failo nusiuntimas ir atsakymo gavimas gali užtrukti žymiai ilgiau nei pats signalo apdorojimas.
5. Kadangi iOS įrenginiai yra mobilūs realiuose taikymuose reikėtų didelį dėmesį skirti aplinkos triukšmų įtakai, nes atpažinimą vykdant prie asmeninio stacionaraus kompiuterio galima gan lengvai izoliuoti pašalinius triukšmus pvz. uždarant langą į gatvę ar užsidarant duris, kai tuo tarpu naudojantis mobiliu įrenginiu pvz. gatvėje pilna aplinkos triukšmų ir pašalinių žmonių balsų, kas smarkiai įtakoja atpažinimo tikslumą. Į šią triukšmų problemą realiuose taikymuose turėtų būti skiriama labai daug dėmesio. Verta paminėti, kad iOS turi integruotą triukšmų slopinimą įrašinėjant, tačiau realus atpažinimo tikslumo pokytis naudojant šį integruotą mechanizmą turi būti nustatytas eksperimentais.
6. Su kiekviena nauja iOS įrenginių procesorių karta pasiekiami vis geresnį atpažinimo greičio rezultatai, jau dabar iPad Air vieną sekundę įrašo vidutiniškai analizuoja vos 25 ms su Julius biblioteka ir 46 ms su CMU Sphinx biblioteka. Tai reiškia, kad jau dabar su iPad Air trumpus kelių sekundžių įrašus galime atpažinti realiu laiku ir tam nereikia interneto prieigos ir serverio. Tas leidžia daryti prielaidą, kad vis daugiau uždavinių bus sprendžiama klientinėje (įrenginio) dalyje, ne išimtis ir lietuvių kalbos atpažinimas. Taigi sistemos galinčios be interneto prieigos atpažinti lietuvių kalbą realiu laiku poreikis tik didės, nes nauji įrenginiai įgalins vis didesnio žodyno kalbos atpažinimą realiu laiku.

LITERATŪRA

[1] Likit, Balso technologijų panaudojimas neįgaliesiems. Likit, [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.likit.lt/all/balso_tech/06_neigalieji.htm

[2] Penki, Technologijos leis kompiuteriais naudotis ir neįgaliesiems. Penki, 2000-02-21 [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.penki.lt/LT/article.im?id=91532
[3] Micheal A. Grasso, Automated Speech Recognition in Medical Applications. Pensilvanija: Penn State University, 1995, 4 p.
[4] Tomas Pocius, Balso atpažinimo sistemos. InfoBitas, 2011-03-16 [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.infobitas.lt/index.php?option=com_content&view=article&id=74
[5] Satoshi Nakamura, Overcoming the Language Barrier with Speech Translation Technology. Quarterly Review No. 31 / April 2009, 38 p.
[6] Agam Shah, Use Voice, Gestures to Control TV. PCWorld 2012-04-04 [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.pcworld.com/article/253223/use_voice_gestures_to_control_tv.html
[7] Vlad Bobleanta, LG adds voice control to its washing machine management smartphone app. UnwiredView, 2012-09-04 [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.unwiredview.com/2012/09/04/lg-adds-voice-control-to-its-washing-machine-management-smartphone-app/
[8] Microsoft, Use Kinect voice commands. Microsfoft [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://support.xbox.com/en-US/xbox-360/kinect/control-your-xbox-360-with-your-voice
[9] Joel Durham, Say2Play. PCMag [interaktyvus] [žiūrėta 2012-12-09]. Prieiga per internetą http://www.pcmag.com/article2/0,2817,2340420,00.asp
[10] Edgar Alvarez, Skyrim gets Kinect integration on Xbox, over 200 voice commands in tow (video). engadget 2012-04-12 [interaktyvus] [žiūrėta 2012-12-08]. Prieiga per internetą http://www.engadget.com/2012/04/12/skyrim-kinect-integration-xbox/
[11] Sheila McGee-Smith, Language (not Speech) in the Contact Center. NoJitter, 2012-04-10 [interaktyvus] [žiūrėta 2012-12-09]. Prieiga per internetą http://www.nojitter.com/post/232900091/language-not-speech-in-the-contact-center
[12] Gal Reuven, Sharon Gannot, Israel Cohen, Joint noise reduction and acoustic echo cancellation using the transfer-function generalized sidelobe canceller. Elsevier, 2007, 623 p.
[13] L. Barcaroli, G. Linares, J.-P. Costa, J.-F. Bonastre, NONLINEAR GSM ECHO CANCELLATION : APPLICATION TO SPEECH RECOGNITION. University of Avignon, 2003, 1-4 p.
[14] A.G. Maher, R.W. King, J.G. Rathmell, Comparison of Noise Reduction Techniques for Speech Recognition in Telecommunications Environments. The Institution of Engineers Australia Communications Conference Sydney, 1992, 109-110 p.
[15] C. Kermorvant, Christopher Kermorvant, A Comparison of Noise Reduction Techniques for Robust Speech Recognition. CiteSeer, 1999 [interaktyvus] [žiūrėta 2012-12-09]. Prieiga per internetą http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.1091
[16] Mark Stamp, A Revealing Introduction to Hidden Markov Models, Department of Computer Science San Jose State University, 2012-09-28, 1-9 p.
[17] Lawrence R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition , IEEE, 1989, 257-265 p.
[18] Gintautas Tamulevičius, Pavienių žodžių atpažinimo sistemų kūrimas, Vilniaus Gedimino technikos universitetas, Matematikos ir informatikos institutas, 2008, 48-49 p.
[19] Lawrence R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition , IEEE, 1989, 264-265 p.
[20] XIE Lingyun, DU Limin, Efficient Viterbi beam search algorithm usin dynamic pruning, Labs for Speech Interaction Technology Research, Institute of Acoustics, Chinese Academy of Sciences, Beijing China, 2004, 699-702 p.
[21] Kęstutis Driaunys, Lietuvių šnekamosios kalbos segmentavimo ir fonetinio atpažinimo tyrimas naudojant LTDIGITS garsyno įrašus, Vilniaus Universitetas, 2006, 32-36 p.
[22] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, Jenifer C. Lai, Class-Based n-gram Models of Natural Language, 1992, 467-470 p.
[23] Juozas Kamarauskas, Asmens atpažinimas pagal balsą, VGTU leidykla TECHNIKA, 2009, 53-55 p.
[24] Katariina Mahkonen, Mel-frequency cepstral coefficients (MFCCs), 2013 [interaktyvus] [žiūrėta 2014-04-25] Prieiga per internetą http://www.cs.tut.fi/kurssit/SGN-14006/PDF/S04-MFCC.pdf
[25] Janie Chang, Speech Recognition Leaps Forward.Microsoft Research, 2011-08-29 [interactyvus]

[žiūrėta 2012-10-21]. Prieiga per internetą http://research.microsoft.com/en-us/news/features/speechrecognition-082911.aspx
[26] Daniel Rios, Neuro AI, [interaktyvus] [žiūrėta 2014-03-20]. Prieiga per internetą http://www.learnartificialneuralnetworks.com/introduction-to-neural-networks.html
[27] Bekir Karlik and A. Vehbi Olgac, Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks, International Journal of Artificial Intelligence And Expert Systems (IJAE), Volume (1): Issue (4), 2010, 111-122 p.
[28] Andrej Krenker, Janez Bešter, Andrej Kos, Artificial Neural Networks – Methodological Advances and Biomodical Applications, InTech, 2011, 1-13 p.
[29] Andrej Krenker, Janez Bešter, Andrej Kos, Artificial Neural Networks – Methodological Advances and Biomodical Applications, InTech, 2011, 14 p.
[30] Gintautas Dzemyda, Olga Kurasova, Julius Žilinskas, Daugiamačių duomenų vizualizavimo metodai, Mokslo Aidai, 2008, 116-118 p.
[31] M. Muller, Information Retrieval for Music and Motion, Springer, 2007, 69-74 p.
[32] CMUSphinx, Training Acoustic Model For CMUSphinx [interaktyvus] [žiūrėta 2014-01-20]. Prieiga per internetą http://cmusphinx.sourceforge.net/wiki/tutorialam
[33] VoxForge, Tutorial: Create Acoustic Model – Manually [interaktyvus] [žiūrėta 2014-01-10]. Prieiga per internetą http://www.voxforge.org/home/dev/acousticmodels/linux/create/htkjulius/tutorial
[34] G. Dahl, Dong Yu, Li Deng, and Alex Acero, Large Vocabulary Continuous Speech Recognition With Context-Dependent DBN-HMMS, IEEE, 2011, 4688-4691 p.

PRIEDAI

1. CMU SPHINX KONFIGŪRACINIS FAILAS

```
# Configuration script for sphinx trainer                  -*-mode:Perl-*-

$CFG_VERBOSE = 1;           # Determines how much goes to the screen.

# These are filled in at configuration time
$CFG_DB_NAME = "sphinx";
# Experiment name, will be used to name model files and log files
$CFG_EXPTNAME = "$CFG_DB_NAME";

# Directory containing SphinxTrain binaries
$CFG_BASE_DIR = "/Users/Jamagas/Documents/Eclipse/LSRE-Tool/Result/Sphinx";
$CFG_SPHINXTRAIN_DIR = "/usr/local/lib/sphinxtrain";
$CFG_BIN_DIR = "/usr/local/libexec/sphinxtrain";
$CFG_SCRIPT_DIR = "/usr/local/lib/sphinxtrain/scripts";


# Audio waveform and feature file information
$CFG_WAVFILES_DIR = "$CFG_BASE_DIR/wav";
$CFG_WAVFILE_EXTENSION = 'wav';
$CFG_WAVFILE_TYPE = 'raw'; # one of nist, mswav, raw
$CFG_FEATFILES_DIR = "$CFG_BASE_DIR/feat";
$CFG_FEATFILE_EXTENSION = 'mfc';
$CFG_VECTOR_LENGTH = 13;

# Feature extraction parameters
$CFG_WAVFILE_SRATE = 16000.0;
$CFG_NUM_FILT = 31; # For wideband speech it's 40, for telephone 8khz reasonable value is 31
$CFG_LO_FILT = 200; # For telephone 8kHz speech value is 200
$CFG_HI_FILT = 3500; # For telephone 8kHz speech value is 3500

$CFG_MIN_ITERATIONS = 1;  # BW Iterate at least this many times
$CFG_MAX_ITERATIONS = 10; # BW Don't iterate more than this, somethings likely wrong.

# (none/max) Type of AGC to apply to input files
$CFG_AGC = 'none';
# (current/none) Type of cepstral mean subtraction/normalization
# to apply to input files
$CFG_CMN = 'current';
# (yes/no) Normalize variance of input files to 1.0
$CFG_VARNORM = 'no';
# (yes/no) Train full covariance matrices
$CFG_FULLVAR = 'no';
# (yes/no) Use diagonals only of full covariance matrices for
# Forward-Backward evaluation (recommended if CFG_FULLVAR is yes)
$CFG_DIAGFULL = 'no';

# (yes/no) Perform vocal tract length normalization in training.  This
# will result in a "normalized" model which requires VTLN to be done
# during decoding as well.
$CFG_VTLN = 'no';
# Starting warp factor for VTLN
$CFG_VTLN_START = 0.80;
# Ending warp factor for VTLN
$CFG_VTLN_END = 1.40;
# Step size of warping factors
$CFG_VTLN_STEP = 0.05;

# Directory to write queue manager logs to
$CFG_QMGR_DIR = "$CFG_BASE_DIR/qmanager";
# Directory to write training logs to
$CFG_LOG_DIR = "$CFG_BASE_DIR/logdir";
# Directory for re-estimation counts
$CFG_BWACCUM_DIR = "$CFG_BASE_DIR/bwaccumdir";
# Directory to write model parameter files to
$CFG_MODEL_DIR = "$CFG_BASE_DIR/model_parameters";

# Directory containing transcripts and control files for
# speaker-adaptive training
$CFG_LIST_DIR = "$CFG_BASE_DIR/etc";

# Decoding variables for MMIE training
```

```
$CFG_LANGUAGEWEIGHT = "11.5";
$CFG_BEAMWIDTH      = "1e-100";
$CFG_WORDBEAM       = "1e-80";
$CFG_LANGUAGEMODEL  = "$CFG_LIST_DIR/$CFG_DB_NAME.lm.DMP";
$CFG_WORDPENALTY    = "0.2";

# Lattice pruning variables
$CFG_ABEAM              = "1e-50";
$CFG_NBEAM              = "1e-10";
$CFG_PRUNED_DENLAT_DIR  = "$CFG_BASE_DIR/pruned_denlat";

# MMIE training related variables
$CFG_MMIE = "no";
$CFG_MMIE_MAX_ITERATIONS = 5;
$CFG_LATTICE_DIR = "$CFG_BASE_DIR/lattice";
$CFG_MMIE_TYPE   = "rand"; # Valid values are "rand", "best" or "ci"
$CFG_MMIE_CONSTE = "3.0";
$CFG_NUMLAT_DIR  = "$CFG_BASE_DIR/numlat";
$CFG_DENLAT_DIR  = "$CFG_BASE_DIR/denlat";

# Variables used in main training of models
$CFG_DICTIONARY     = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";
$CFG_RAWPHONEFILE   = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";
$CFG_FILLERDICT     = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";
$CFG_LISTOFFILES    = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";
$CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription";
$CFG_FEATPARAMS     = "$CFG_LIST_DIR/feat.params";

# Variables used in characterizing models

$CFG_HMM_TYPE = '.cont.'; # Sphinx 4, PocketSphinx
#$CFG_HMM_TYPE  = '.semi.'; # PocketSphinx
#$CFG_HMM_TYPE  = '.ptm.'; # PocketSphinx (larger data sets)

if (($CFG_HMM_TYPE ne ".semi.")
    and ($CFG_HMM_TYPE ne ".ptm.")
    and ($CFG_HMM_TYPE ne ".cont.")) {
  die "Please choose one CFG_HMM_TYPE out of '.cont.', '.ptm.', or '.semi.', " .
    "currently $CFG_HMM_TYPE\n";
}

# This configuration is fastest and best for most acoustic models in
# PocketSphinx and Sphinx-III.  See below for Sphinx-II.
$CFG_STATESPERHMM = 3;
$CFG_SKIPSTATE = 'no';

if ($CFG_HMM_TYPE eq '.semi.') {
  $CFG_DIRLABEL = 'semi';
# Four stream features for PocketSphinx
  $CFG_FEATURE = "s2_4x";
  $CFG_NUM_STREAMS = 4;
  $CFG_INITIAL_NUM_DENSITIES = 256;
  $CFG_FINAL_NUM_DENSITIES = 256;
  die "For semi continuous models, the initial and final models have the same density"
    if ($CFG_INITIAL_NUM_DENSITIES != $CFG_FINAL_NUM_DENSITIES);
} elsif ($CFG_HMM_TYPE eq '.ptm.') {
  $CFG_DIRLABEL = 'ptm';
# Four stream features for PocketSphinx
  $CFG_FEATURE = "s2_4x";
  $CFG_NUM_STREAMS = 4;
  $CFG_INITIAL_NUM_DENSITIES = 64;
  $CFG_FINAL_NUM_DENSITIES = 64;
  die "For phonetically tied models, the initial and final models have the same density"
    if ($CFG_INITIAL_NUM_DENSITIES != $CFG_FINAL_NUM_DENSITIES);
} elsif ($CFG_HMM_TYPE eq '.cont.') {
  $CFG_DIRLABEL = 'cont';
# Single stream features - Sphinx 3
  $CFG_FEATURE = "1s_c_d_dd";
  $CFG_NUM_STREAMS = 1;
  $CFG_INITIAL_NUM_DENSITIES = 1;
  $CFG_FINAL_NUM_DENSITIES = 8;
  die "The initial has to be less than the final number of densities"
    if ($CFG_INITIAL_NUM_DENSITIES > $CFG_FINAL_NUM_DENSITIES);
}

# Number of top gaussians to score a frame. A little bit less accurate computations
# make training significantly faster. Uncomment to apply this during the training
# For good accuracy make sure you are using the same setting in decoder
```

```
# In theory this can be different for various training stages. For example 4 for
# CI stage and 16 for CD stage
# $CFG_CI_TOPN = 4;
# $CFG_CD_TOPN = 16;

# (yes/no) Train multiple-gaussian context-independent models (useful
# for alignment, use 'no' otherwise) in the models created
# specifically for forced alignment
$CFG_FALIGN_CI_MGAU = 'no';
# (yes/no) Train multiple-gaussian context-independent models (useful
# for alignment, use 'no' otherwise)
$CFG_CI_MGAU = 'no';
# Number of tied states (senones) to create in decision-tree clustering
$CFG_N_TIED_STATES = 200;
# How many parts to run Forward-Backward estimatinon in
$CFG_NPART = 1;

# (yes/no) Train a single decision tree for all phones (actually one
# per state) (useful for grapheme-based models, use 'no' otherwise)
$CFG_CROSS_PHONE_TREES = 'no';

# Use force-aligned transcripts (if available) as input to training
$CFG_FORCEDALIGN = 'no';

# Use a specific set of models for force alignment.  If not defined,
# context-independent models for the current experiment will be used.
$CFG_FORCE_ALIGN_MDEF = "$CFG_BASE_DIR/model_architecture/$CFG_EXPTNAME.falign_ci.mdef";
$CFG_FORCE_ALIGN_MODELDIR = "$CFG_MODEL_DIR/$CFG_EXPTNAME.falign_ci_$CFG_DIRLABEL";

# Use a specific dictionary and filler dictionary for force alignment.
# If these are not defined, a dictionary and filler dictionary will be
# created from $CFG_DICTIONARY and $CFG_FILLERDICT, with noise words
# removed from the filler dictionary and added to the dictionary (this
# is because the force alignment is not very good at inserting them)

# $CFG_FORCE_ALIGN_DICTIONARY = "$ST::CFG_BASE_DIR/falignout$ST::CFG_EXPTNAME.falign.dict";;
# $CFG_FORCE_ALIGN_FILLERDICT = "$ST::CFG_BASE_DIR/falignout/$ST::CFG_EXPTNAME.falign.fdict";;

# Use a particular beam width for force alignment.  The wider
# (i.e. smaller numerically) the beam, the fewer sentences will be
# rejected for bad alignment.
$CFG_FORCE_ALIGN_BEAM = 1e-60;

# Calculate an LDA/MLLT transform?
$CFG_LDA_MLLT = 'no';
# Dimensionality of LDA/MLLT output
$CFG_LDA_DIMENSION = 29;

# This is actually just a difference in log space (it doesn't make
# sense otherwise, because different feature parameters have very
# different likelihoods)
$CFG_CONVERGENCE_RATIO = 0.1;

# Queue::POSIX for multiple CPUs on a local machine
# Queue::PBS to use a PBS/TORQUE queue
$CFG_QUEUE_TYPE = "Queue";

# Name of queue to use for PBS/TORQUE
$CFG_QUEUE_NAME = "workq";

# (yes/no) Build questions for decision tree clustering automatically
$CFG_MAKE_QUESTS = "yes";
# If CFG_MAKE_QUESTS is yes, questions are written to this file.
# If CFG_MAKE_QUESTS is no, questions are read from this file.
$CFG_QUESTION_SET = "${CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.tree_questions";
#$CFG_QUESTION_SET = "${CFG_BASE_DIR}/linguistic_questions";

$CFG_CP_OPERATION = "${CFG_BASE_DIR}/model_architecture/${CFG_EXPTNAME}.cpmeanvar";

# Configuration for grapheme-to-phoneme model
$CFG_G2P_MODEL= 'no';

# Configuration script for sphinx decoder

# Variables starting with $DEC_CFG_ refer to decoder specific
# arguments, those starting with $CFG_ refer to trainer arguments,
# some of them also used by the decoder.
```

```
$DEC_CFG_VERBOSE = 1;       # Determines how much goes to the screen.

# These are filled in at configuration time

# Name of the decoding script to use (psdecode.pl or s3decode.pl, probably)
$DEC_CFG_SCRIPT = 'psdecode.pl';

$DEC_CFG_EXPTNAME = "$CFG_EXPTNAME";
$DEC_CFG_JOBNAME  = "$CFG_EXPTNAME"."_job";

# Models to use.
$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";

$DEC_CFG_FEATFILES_DIR = "$CFG_BASE_DIR/feat";
$DEC_CFG_FEATFILE_EXTENSION = '.mfc';
$DEC_CFG_VECTOR_LENGTH = $CFG_VECTOR_LENGTH;
$DEC_CFG_AGC = $CFG_AGC;
$DEC_CFG_CMN = $CFG_CMN;
$DEC_CFG_VARNORM = $CFG_VARNORM;

$DEC_CFG_QMGR_DIR = "$CFG_BASE_DIR/qmanager";
$DEC_CFG_LOG_DIR = "$CFG_BASE_DIR/logdir";
$DEC_CFG_MODEL_DIR = "$CFG_MODEL_DIR";

$DEC_CFG_DICTIONARY     = "$CFG_BASE_DIR/etc/$CFG_DB_NAME.dic";
$DEC_CFG_FILLERDICT     = "$CFG_BASE_DIR/etc/$CFG_DB_NAME.filler";
$DEC_CFG_LISTOFFILES    = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.fileids";
$DEC_CFG_TRANSCRIPTFILE = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.transcription";
$DEC_CFG_RESULT_DIR     = "$CFG_BASE_DIR/result";

# This variables, used by the decoder, have to be user defined, and
# may affect the decoder output

$DEC_CFG_LANGUAGEMODEL  = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}.lm.DMP";
$DEC_CFG_LANGUAGEWEIGHT = "10";
$DEC_CFG_BEAMWIDTH = "1e-80";
$DEC_CFG_WORDBEAM = "1e-40";

$DEC_CFG_ALIGN = "builtin";

$DEC_CFG_NPART = 1;         #  Define how many pieces to split decode in

# This variable has to be defined, otherwise utils.pl will not load.
$CFG_DONE = 1;

return 1;
```

2. HTK KONFIGŪRACINIS FAILAS

```
TARGETKIND = MFCC_0_D_N_Z
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
```

3. CSV FAILO PAVYZDYS

```
Julius,astuoni-101983.wav,1.209375,0.1404749751091003,astuoni,ASTUONI,1
Julius,keturi-101423.wav,1.059375,0.1225699782371521,keturi,KETURI,1
Julius,keturi-101424.wav,1.096875,0.1280320286750793,keturi,PENKI,0
Julius,trys-101124.wav,0.984375,0.112962007522583,trys,TRYS,1
Julius,trys-101140.wav,0.796875,0.08969098329544067,trys,DU,0
Julius,trys-101141.wav,0.834375,0.09540897607803345,trys,PENKI,0
<...>
```

4. JULIUS FAILŲ KOREKCIJOS

Libsent/src/anlz/vecin_net.c

```
218 +     jlog("Error: vecin_net: vecin_get_configuration() function end reached\n");
219 +     return 0;
```

libsent/src/phmm/outprob.c

```
472 +     return FALSE;
```