Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Skaičių garsyno atpažinimo su Kaldi paketu sistemos sukūrimas ir tyrimas

Baigiamasis magistro projektas

Giedrius Sinkevičius

Projekto autorius

Doc. dr. Kastytis Ratkevičius

Vadovas

Kaunas, 2019

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Skaičių garsyno atpažinimo su Kaldi paketu sistemos sukūrimas ir tyrimas

Baigiamasis magistro projektas

Valdymo technologijos (6211EX014)

Giedrius Sinkevičius

Projekto autorius

Doc. dr. Kastytis Ratkevičius

Vadovas

Doc. dr. Vytautas Gargasas

Recenzentas

Kaunas, 2019

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Giedrius Sinkevičius

Skaičių garsyno atpažinimo su Kaldi paketu sistemos sukūrimas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Giedriaus Sinkevičiaus, baigiamasis projektas tema „Skaičių garsyno atpažinimo su Kaldi paketu sistemos sukūrimas ir tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

_____	_____
(vardą ir pavardę įrašyti ranka)	(parašas)

Sinkevičius, Giedrius. Skaičių garsyno atpažinimo su Kaldi paketu sistemos sukūrimas ir tyrimas. Magistro baigiamasis projektas / vadovas doc. dr. Kastytis Ratkevičius; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, krypčių grupė – inžinerijos mokslai.

Reikšminiai žodžiai: Kaldi, paslėptas Markovo modelis, dirbtinis neuroninis tinklas, akustinis modeliavimas

Kaunas, 2019. 60 p.

Santrauka

Darbo tikslas - išnagrinėti *Kaldi* paketo pritaikymo ir funkcijų galimybes automatinio kalbos atpažinimo sistemose. Eksperimentams buvo pasirinktas plačiausiai naudojamas garso įrašų rinkinys *SKAIC30* – 30 diktorių balso įrašai su lietuviškų skaičių nuo 0 iki 9 komandų ištarimais, kuris buvo išplėstas iki 100 diktorių balso įrašų su 5dB foniniu triukšmu. Baigiamajame darbe pateikiamas palyginimas su HTK programinio paketo gaunamais rezultatais, naudojant 30 diktorių balso įrašus. Tolimesniems tyrimams naudojamas 100 diktorių garsynas su foniniu 5dB triukšmu, nagrinėjama, kuriuo kalbos atpažinimo metodu – monofoninu, trifoniniu, *LDA+MLLT*, *LDA+MLLT+SAT*, *SGMM*, *DNN* - gaunama mažiausia atpažinimo paklaida.

Bandymams paruošti *Kaldi* aprašomieji garsyno failai: *spk2gender.txt, wav.scp, text.txt, utt2spk.txt, corpus.txt*, kurie reikalingi sistemos apmokymui ir testavimui, taip pat pagrindinis *run.sh* failas, kuriame aprašomos naudojamos funkcijos.

Atlikus pirminius bandymus su 30 diktorių garsynu paaiškėjo, kad sistemos atpažinimo tikslumas, naudojant paprasčiausią monofoninį metodą yra panašus į atpažinimo sistemos, sukurtos su *HTK* programiniu paketu, tikslumą. Remiantis [1] straipsniu, su naudojamu tokiu pat *SKAIC30* diktorių garsynu kalbos atpažintuvas pasiekė 99 % atpažinimo tikslumą. Tuo tarpu šiame darbe sistemos, sukurtos su Kaldi programiniu paketu geriausias bandymų rezultatas siekia 99,75 % tikslumą.

Naudojant kiekvieną iš prieš tai paminėtų metodų buvo atliktas kryžminis patikrinimas su 100 diktorių garsynu taip, kad kiekvienas diktorius pakliūtų į testavimo ir apmokymo etapus. Didžiausias kalbos atpažinimo tikslumas buvo gautas naudojant *DNN* akustinio modeliavimo metodą (98.2 %).

Sinkevičius, Giedrius. Creation and Investigation of Digits Speech Corpus Recognition System Using Kaldi Package: Master‘s degree / supervisor assoc. doc. dr. Kastytis Ratkevičius; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): electronics engineering, engineering science;

Keywords: automatic speech recognition, kaldi toolkit, hidden markov model, deep neural network

Kaunas, 2019. 60 pages.

Summary

Purpose of this project is to analyze Kaldi toolkit possibilities in automatic speech recognition researches. The most widely audio collection was used, named *SKAIC30* - 30 speakers voice recordings with Lithuanian numbers from 0 to 9, which was expanded to 100 speakers voice recordings with 5dB background noise. The final work presents a comparison with the results of the *HTK* software package using 30 voice recorders. For further research, a 100 speakers voice recordings with 5dB background noise was used, in order to check which method of the following: *monophone*, *triphone*, *LDA+MLLT*, *LDA+MLLT+SAT*, *SGMM*, *DNN* provides the most accurate results for automatic speech recognition system.

Kaldi descriptive sound files: *spk2gender.txt*, *wav.scp*, *text.txt*, *utt2spk.txt*, *corpus.txt* were generated for research, which are required for system training and testing, as well as the main run.sh file, describing the functions used.

Initial testing results with the 30 speakers voice recordings revealed that the accuracy of automatic speech recognition system using a simple monophonic method is similar to automatic speech recognition system accuracy created with HTK software. According to Article [1], researches were made with the same *SKAIC30* voice recordings and system has reached 99% accuracy. Meanwhile, in this work, the system created with the Kaldi software package has the best test result of 99.75% accuracy.

Using each of the above-mentioned methods, a cross-check with 100 speakers voice recordings was performed so that each speaker would be used in testing and training stages. The highest accuracy of speech recognition was obtained using the DNN acoustic modeling method (98.2 %).

Turinys

Lentelių sąrašas **8**
Paveikslų sąrašas **9**
Santrumpų sąrašas **11**
Įvadas **12**
1. Kalbos atpažinimas **14**
1.1. Kalbos atpažinimui naudojamos programos 14
1.1.1. HTK paketas 14
1.1.2. CMUSphinx paketas 15
1.1.3. Julius paketas 15
1.1.4. Kaldi paketas 16
1.2. Kalbos atpažintuvo kūrimo uždaviniai 16
1.2.1. Požymių išskyrimas 16
1.2.2. Melų dažnių skalės kepstrinis koeficientas (MFCC) 17
1.2.3. Dekodavimas 18
1.2.4. Akustinis modeliavimas 18
1.2.5. Kalbos modelis 19
1.3. Akustinio modeliavimo metodai 19
1.3.1. Paslėptas Markovo modelis 19
1.3.2. Monofoninis metodas 21
1.3.3. Trifoninis metodas 21
1.3.4. Linijinė diskriminanto analizė (LDA) 21
1.3.5. Prisitaikantis diktoriaus mokymas (SAT) 22
1.3.6. Tarpinis Gauso mišinių modelis (SGMM) 22
1.3.7. Neuroniniai tinklai (DNN) 23
1.3.8. Neuroninio tinklo apmokymas kalbos atpažinimo sistemoje 23
1.4. Kaldi programinio paketo apžvalga 24
1.4.1. Automatinio kalbos atpažinimo sistemos dalys Kaldi programiniame pakete 24
2. Metodinė dalis **26**
2.1. Darbinės aplinkos paruošimas 26
2.1.1. Ubuntu sistemos įdiegimas 26
2.1.2. Papildomų bibliotekų įrašymas 26
2.1.3. Kaldi programinio paketo įdiegimas 27
2.2. Garsyno aprašymas 28
2.2.1. 30 diktorių garsynas 28
2.2.2. 100 diktorių garsynas 28
2.3. Automatinio kalbos atpažintuvo kūrimas su Kaldi programiniu paketu 29
2.3.1. Projekto sukūrimas 29
2.3.2. Garsyno paruošimas projektui 29
2.3.3. Fonemų aprašomieji failai 33
2.3.4. Papildomi konfigūraciniai failai 34
2.3.5. Pagrindinis run.sh skripto aprašymas 36
3. Rezultatai **43**

3.1. 30 diktorių garsyno bandymų rezultatai 43
3.2. 100 diktorių garsyno su foniniu 5 dB triukšmu bandymų rezultatai 45
3.3. Bendras 6 metodų palyginimas. 53
3.4. Tyrimai su SAMPA-LT fonemų rinkiniu 54
Išvados 58
Literatūros sąrašas 59
Priedai 61
1 priedas. Java programos kodas aprašomiesiems garsyno failams generuoti 61

Lentelių sąrašas

1 lentelė. 30 diktorių garsyno atpažinimo rezultatai. Monofoninis metodas..... 44
2 lentelė. Gautų bandymų rezultatų vidurkiai, geriausias, prasčiausias rezultatas..... 45
3 lentelė. Kryžminio bandymo rezultatai, pritaikius monofoninį akustinį modelį. 46
4 lentelė. Bandymų su monofoniniu akustiniu modeliu rezultatų suvestinė. 46
5 lentelė. Kryžminio bandymo rezultatai, pritaikius trifoninį akustinį modelį. 47
6 lentelė. Bandymų su trifoniniu akustiniu modeliu rezultatų suvestinė..... 47
7 lentelė. Kryžminio bandymo rezultatai, pritaikius *LDA+MLLT* akustinį modelį. 48
8 lentelė. Bandymų su LDA+MLLT akustiniu modeliu rezultatų suvestinė. 49
9 lentelė. . Kryžminio bandymo rezultatai, pritaikius *LDA+MLLT+SAT* akustinį modelį..... 50
10 lentelė. Bandymų su *LDA+MLLT+SAT* akustiniu modeliu rezultatų suvestinė. 50
11 lentelė. Kryžminio bandymo rezultatai, pritaikius *SGMM* akustinį modelį. 51
12 lentelė. Bandymų su *SGMM* akustiniu modeliu rezultatų suvestinė. 51
13 lentelė. Kryžminio bandymo rezultatai, pritaikius *DNN* akustinį modelį 52
14 lentelė. Bandymų su *DNN* akustiniu modeliu rezultatų suvestinė..... 52
15 lentelė. *B-aplanko* bandymų rezultatai su *SAMPA-LT* fonemų rinkiniu, panaudojant 6 skirtingus modeliavimo metodus. 56

Paveikslų sąrašas

1 pav. Principinė kalbos atpažinimo schema. ... 14
2 pav. Automatinio kalbos atpažintuvo schema. ... 16
3 pav. MFCC išvestis. ... 18
4 pav. Markovo grandinių struktūros. (a) ergodinis modelis, (b) Bakio modelis. [9] ... 20
5 pav. Duomenų rinkiniai, atvaizduoti originalioje erdvėje ir transformuotoje erdvėje [11] ... 21
6 pav. *SAT* modelis konkretaus diktoriaus kalbėsenos modeliavimui[13]. ... 22
7 pav. Kaldi programinio įrankio struktūra ir naudojamos bibliotekos [18]. ... 24
8 pav. Disko skaidymo langas. ... 26
9 pav. *Check_dependencies.sh* failo kodo fragmentas. ... 27
10 pav. *Git* bibliotekos komanda, naudojama Kaldi programiniam paketui parsiųsti iš *GitHub* repositorijos. ... 27
11 pav. Komandinės eilutės skirtos *Kaldi* programiniam paketui sukompiliuoti. ... 28
12 pav. *Visual studio code* editoriaus langas su atidarytu *sk_24_6* projektu. ... 29
13 pav. *Spk2gender* failo fragmentas. ... 30
14 pav. *Utt2spk* failo turinio fragmentas. ... 30
15 pav. *Text* failo turinio fragmentas ... 31
16 pav. *Wav.scp* failo turinio fragmentas. ... 31
17 pav. Garsyno informacinių failų tarpusavio sąsaja. ... 32
18 pav. *Lexicon.txt* failo turinys. ... 33
19 pav. *Nonsilence_phones.txt.* Tariamų fonemų (grafemų) sąrašas ... 34
20 pav. *Silence_phone.txt*. Tylos fonemų sąrašas ... 34
21 pav. *Optional_silence.txt*. Papildomų tylos fonemų sąrašas. ... 34
22 pav. *Mfcc.conf* konfigūracinio failo turinys. ... 35
23 pav. *Decode.config* konfigūracinio failo turinys ... 35
24 pav. *Path.sh* konfigūracinio failo turinys ... 35
25 pav. *Cmd.sh* konfigūracinio failo turinys ... 35
26 pav. Konfigūracinių nustatymų skriptas. ... 36
27 pav. Garsyno failų paruošimo skriptas ... 36
28 pav. Požymių išskyrimo skriptas ... 36
29 pav. Kalbos modelio skriptas ... 37
30 pav. Monofoninio modelio skriptas ... 37
31 pav. Monofoninio modelio testavimo skriptas ... 37
32 pav. *Linux* terminalo langas ... 38
33 pav. Monofoninio modelio paruošimo trifoniniam metodui skriptas ... 38
34 pav. Trifoninio modelio skriptas ... 38
35 pav. Trifoninio modelio testavimo skriptas ... 39
36 pav. Trifoninio modelio paruošimo skriptas ... 39
37 pav. LDA+MLLT modelio skriptas ... 39
38 pav. LDA+MLLT modelio testavimo skriptas ... 39
39 pav. LDA+MLLT+SAT modelio skriptas ... 40
40 pav. LDA+MLLT+SAT modelio testavimo skriptas ... 40
41 pav. SGMM modelio skriptas ... 40
42 pav. SGMM modelio testavimo skriptas ... 41
43 pav. DNN modelio skriptas ... 41

44 pav. DNN modelio testavimo skriptas 42
45 pav. *Linux* terminalo langas su pateiktais bandymų rezultatais. 43
46 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant monofoninį akustinį modelį 47
47 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant trifoninį akustinį modelį..... 48
48 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *LDA+MLLT* akustinį modelį 49
49 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *LDA+MLLT+SAT* akustinį modelį 50
50 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *SGMM* akustinį modelį 52
51 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *DNN* akustinį modelį 53
52 pav. 6 metodų bandymų rezultatai viename grafike. 54
53 pav. *SAMPA-LT* fonemų rinkinio sąrašas 55
54 pav. *lexicon.txt* failas aprašytas panaudojant *SAMPA-LT* fonemų rinkinį 56
55 pav. Skirtingų fonemų rinkinių bandymų rezultatai. (1 blokas – *monofinis metodas*, 2 blokas – *trifoninis metodas*, 3 blokas – *LDA+MLLT*, 4 blokas – *LDA+MLLT+SAT*, 5 blokas – *SGMM*, 6 blokas – *DNN*) 57

Santrumpų sąrašas

Santrumpos:

Doc. – docentas;

Dr. – daktaras;

HTK – programinis paketas *hidden Markov model toolkit*;

LDA – linijinė diskriminantinė analizė (angl. *linear discriminant analysis*) ;

MLLT – didžiausios tikimybės linijinė transformacija (angl. *maximum likelihood linear transform*);

SAT – prisitaikantis diktoriaus mokymas (angl. *speaker adaptive training*);

fMLLR – požymių būsenų didžiausios tikimybės linijinė regresija (angl. *feature space maximum likelihood linear regression*);

GMM – Gauso mišinių modelis (angl. *Gaussian mixture model*);

SGMM – tarpinis Gauso mišinių modelis (angl. *subspace Gaussian mixture model*);

DNN – gilieji neuroniniai tinklai (angl. *deep neural networks*) ;

BSD – *Berkeley* programinės įrangos platinimas (angl. *Berkeley software distribution)*;

MFCC – Melų dažnių skalės kepstriniai koeficientai (angl. *Mel frequency cepstral coefficient*);

HMM – paslėptas Markovo modelis (angl. *hidden Markov model*);

FST – baigtinių būsenų keitikliai (angl. *finite-state transducers*);

WFST – svertiniai baigtinės būsenos keitikliai (angl. *Weight finite-state transducers*);

OS – operacinė sistema;

JVM – Java virtuali mašina (angl. *Java virtual machine*);

Įvadas

Pastaruoju metu kalbos atpažinimui skiriamas ypatingas dėmesys. Panaudojant tokias technologijas, kaip gilieji neuroniniai tinklai, mašininis mokymas, dirbtinis intelektas sukurta eilė kalbos atpažinimo taikymų: balsinė navigacija, balsinis valdymas automobiliuose ir išmaniuosiuose telefonuose (*Apple* kompanijos programa „*Siri*“, *Samsung* – „*S-voice*“, *Microsoft* – „*Cortana*“), google paieškos pagalbininkas „*Google Assitant*“, *Amazon* sukurtas išmanusis balso pagalbininkas „*Alexa*“, kuris gali atlikti žadintuvo, radijo funkcijas, taip pat valdyti „išmanius namus“ ir kt.

Kalbos atpažinimo technologijos įgalina supaprastinti žmogaus komunikavimą su kompiuteriais, ir kitais elektroniniais įtaisais. Kalbos atpažinimas ypatingai svarbus neįgaliems žmonėms, kadangi tai natūraliausia priemonė įvairių įregimų valdymui balsu.

Deja, didžioji dalis naujausių pasiekimų kalbos technologijų srityje skirta ne lietuvių, o, pvz., anglų, ispanų, kiniečių kalboms. Straipsnyje [2] rašoma, kad *Google* panaudodama mašininį mokymą pasiekė 95 % anglų kalbos atpažinimo tikslumą. Tiesiogiai pritaikyti šių technologijų lietuvių kalbai nėra galimybių, nes reikia įvertinti lietuvių kalbos specifiką.

Šiame darbe pradiniams tyrimams palyginimui su *HTK* paketu naudojamas lietuviškas 30 diktorių skaičių garsynas. Toliau šis garsynas išplečiamas iki 100 diktorių ir naudojami 6 skirtingi metodai šnekos aktui atpažinti.

Darbo tikslas – ištirti *Kaldi* paketo funkcionalumą panaudojant skirtingus metodus kalbos atpažinimui.

Iškelti uždaviniai:

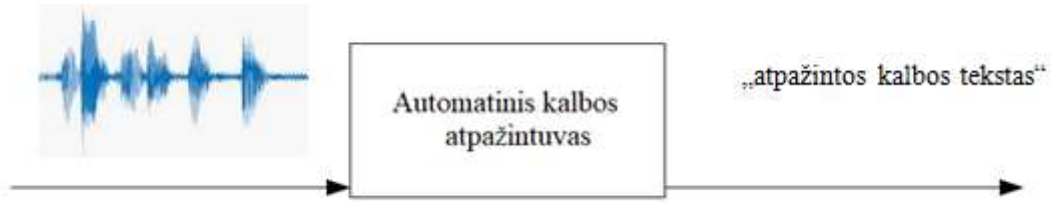
1. Iš literatūros apžvalgos nustatyti svarbiausias kalbos technologijas;
2. išsiaiškinti *Kaldi* programinio paketo funkcijas ir panaudojimo galimybes;
3. apžvelgti pagrindinius kalbos atpažinimo būdus;
4. paruošti kompiuterinę darbo aplinką, tinkamą *Kaldi* programiniam paketui;
5. pritaikyti naudojamus garsynus darbui su *Kaldi* paketu;
6. surasti geriausius atpažinimo metodus;
7. pateikti rezultatus ir parašyti išvadas.

Darbo objektas. Naudojama *Linux* operacinė sistema, Kaldi programinis paketas, du skaičių garsyno variantai. Analizuojami monofoninis ir trifoninis atpažinimo metodai, tiesinė diskriminantinė analizė, adaptyvus kalbėtojui apmokymas ir gilieji neuroniniai tinklai. Taip pat naudojamas užtriukšmintas 5 dB lygyje 100 diktorių skaičių garsynas.

Temos aktualumas. Iki šiol tokio tipo tyrimai buvo atliekami su HTK paketu – tai reikalavo labai daug laiko sąnaudų. Siekiama ištirti specializuoto Kaldi paketo, skirto, būtent kalbos atpažinimui, galimybes.

1. Kalbos atpažinimas

Automatinio kalbos atpažinimo sistemos skirtos sukurti sąsają tarp žmogaus ir kompiuterio, kuri būtų valdoma balsu. Atpažinimo sistema gautą signalą per mikrofoną ar garso įrašą jį apdoroja, ir šios sistemos išėjime gaunama to signalo tekstinė reprezentacija. „atpažintos kalbos tekstas“



1 pav. Principinė kalbos atpažinimo schema.

Kalbos atpažinimo sistemos gali būti suskirstytos į tris pagrindines kategorijas:

1. priklausomos nuo kalbėtojo, kai sistema yra apmokyta ne tik atpažinti ką sako žmogus, bet ir jį identifikuoti. Tokiose sistemose labiau kreipiamas dėmesys į kalbėtojo atpažinimą, o ne į kalbos atpažinimo tikslumą;
2. nepriklausomos nuo kalbėtojo, kai apmokomieji sistemos duomenys yra sudaryti iš didelio kiekio skirtingų diktorių, norint kad sistema kuo geriau atpažintų, bet kurio žmogaus ištartus žodžius. Šiose sistemose akcentuojamas didelis kalbos atpažinimo tikslumas;
3. prisitaikančios, kai sistema iš pradžių yra sukurta remiantis nepriklausomo kalbėtojo metodika, o paskui, panaudojus papildomus sistemos apmokymus, prisitaiko prie individualaus žmogaus ir geba jį atpažinti.

Šiame darbe bus kuriama ir tiriama nepriklausomos nuo kalbėtojo automatinio kalbos atpažinimo sistema, nes turimas skaičių garsynas yra sudarytas iš skirtingų diktorių ištarimų skirtas apmokyti bei testuoti būtent tokio pobūdžio sistemoms.

1.1. Kalbos atpažinimui naudojamos programos

1.1.1. HTK paketas

HTK programinis paketas skirtas kurti ir manipuliuoti gerai žinomu metodu – *paslėptu Markovo modeliu*. Paketo pagrindinė paskirtis yra kurti kalbos atpažinimo sistemas. Tačiau paminėtina tai, kad jį galima panaudoti kalbos sintezės tyrimuose, bruožų atpažinimuose ir *DNR* sekose. *HTK* įrankis yra parašytas *ANSI C* kalba ir veikia tiek ant *Unix* operacinių sistemų, tiek ant *Windows* ar bet kurios kitos modernios OS. Šiuo metu įrankis naudojamas daugiau kaip 100 kalbos tyrimų

laboratorijų iš viso pasaulio ir taip pat naudojamas mokomiesiems tikslams daugelyje universitetų [3].

Daugybė tyrimų jau yra atlikta naudojantis šiuo programiniu paketu, įskaitant straipsnį [1], kurio rezultatai bus palyginti su šiame darbe atliktų tyrimų rezultatais.

1.1.2. CMUSphinx paketas

Viena populiariausių atvirojo kodo kalbos atpažinimų sistemų yra *CMUSphinx* arba tiesiog *Sphinx*. Programa buvo sukurta *Carnegie'o Mellon'o* universiteto kalbos atpažinimo programinės įrangos kūrimo komandos – *Xuedong Huang*. Programinį įrankį sudaro kalbos atpažintuvų serijos (*Sphinx 2-4*) ir akustinių modelių apmokymo modelis (*Sphinx train*).

Sphinx yra nepriklausomas nuo kalbėtojo ištisinės kalbos atpažintuvas, kuris naudoja *paslėptą Markovo modelį* ir *n-gram* statistinį kalbos modelį.

Yra keletas šio programinio paketo atmainų. Pagal *BSD* licenciją, visas angl. „*source code*" priklauso *BSD*, o visi koregavimai ir plėtiniai – jų autoriams. Vienas iš pavyzdžių - *Sphinx2* - greičiausiai veikianti atmaina, kuri yra pritaikoma realaus laiko sistemose. *Sphinx2* naudojama įvairiose išmaniųjų telefonų programėlėse. Po aktyvių *Sphinx* programos tobulinimo darbų, ir naujų metodų pritaikymų (*LDA/MLLT, MLLR, VTLN*), kurie pagerino programinio paketo atpažinimo tikslumą, dabar tai yra galingas įrankis kuriant automatines kalbos atpažinimo sistemas [4].

1.1.3. Julius paketas

Dar vienas nemokamas, atviro kodo kalbos atpažinimo įrankis, kuris buvo sukurtas specialiai japonų kalbai, tad vienas pagrindinių šio programinio paketų trūkumų yra tas, kad nepalaiko kitų kalbų akustinių modelių, tačiau projektas *VoxForge* stengiasi sukurti anglų kalbos akustinį modelį *Julius* įrankiui.

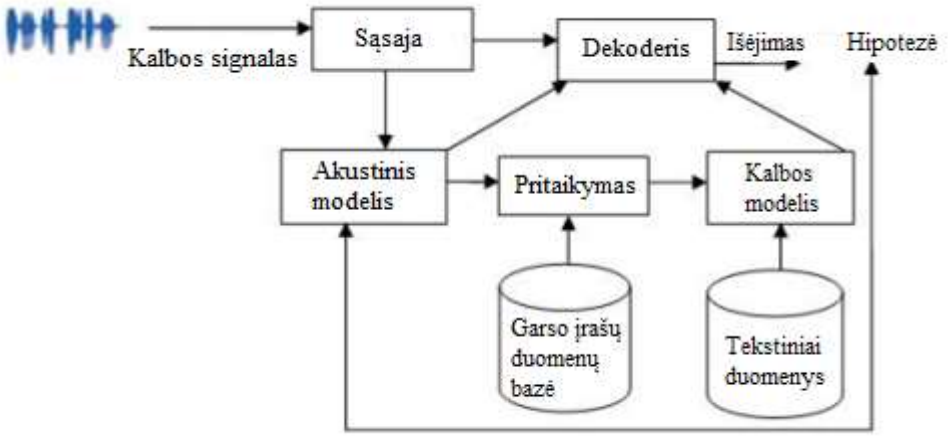
Julius programinis paketas yra greitai atliekanti atpažinimo procesus su dideliu duomenų kiekiu platforma skirta programų kūrėjams bei tyrėjams. Su šia, žodžių *n-gram* ir kontekstiniu *HMM* grįsta programa galima realizuoti realaus laiko atpažintuvą su bet kuriuo kompiuteriu. Čia naudojamos technologijos tokios kaip *grafų leksikonas*, *n-gram* algortimas, *Gauso genėjimas* (angl. *Gaussian prunning*), *Gauso pasirinkimas* (angl. *Gaussian selection*) ir kt. Palaikomi įvairūs *HMM* tipai: *bendros būsenos trifonai*, įvairių kombinacijų *susietų mišinių modeliai*. Išlaikyti bendri programos kūrimo principai bei reikalavimai, kad būtų galima daryti integracijas su kitais kalbos atpažinimo įrankiais, pvz. *HTK*. *Julius* įrankis buvo kurtas *Unix* tipo operacinėms sistemoms, tačiau galima gauti ir *Windows* operacinę sistemą palaikančią programos versiją [5].

1.1.4. Kaldi paketas

Baigiamajame projekte naudojamas *Kaldi* programinis paketas, kurį sudaro *shell* skriptai ir C++ kalba parašytos programos kalbos atpažinimo užduotims vykdyti. Šis programinis paketas sukurtas pagal *Apache v2.0* licenciją, yra lengvai modifikuojamas, pritaikant konkrečioms kalbos atpažinimo užduotims ir vis dar yra kūrimo stadijoje. Taip pat *Kaldi* naudoja dvi pagrindines išorines bibliotekas: *BLAS/LAPACK* biblioteka skirta linijinės algebros skaičiavimams, *OpenFST* skirta atlikti operacijoms su *baigtinių būsenų keitikliais* (angl. *finite state transducers*). Plačiau apie šį programinį paketą aprašyta 1.4 poskyryje.

1.2. Kalbos atpažintuvo kūrimo uždaviniai

Kalbos atpažinimo sistemos susideda iš 4 pagrindinių blokų: požymių išskyrimo, akustinio modeliavimo, kalbos modeliavimo ir dekoderio. Kalbos atpažinimo procesas prasideda nuo kalbėtojo ištarimo, kuris susideda iš garso bangų. Mikrofonas šias garso bangas paverčia elektriniais signalais, kurie paverčiami skaitmeniniais kompiuteriui suprantamais signalais. Tada skaitmeninis signalas yra išskaidomas į diskretinę požymių vektorių seką, kurie turi savyje individualią informaciją, reikalingą ištarimui atpažinti. Galiausiai dekoderis suranda geriausią atitikmenį duomenų bazėje ir dekoderio išėjime matomas rezultatas. Kartais požymių vektorių nešama informacija gali koreliuoti su panašaus skambesio ištarimais, kas sulėtina atpažinimo procesą. Šios problemos optimizavimui, požymių išskyrimui naudojami tokie metodai kaip *Melų dažnių skalės kepstriniai koeficientai* [6].



2 pav. Automatinio kalbos atpažintuvo schema.

1.2.1. Požymių išskyrimas

Požymių išskyrimas kalbos atpažinimo sistemose yra požymių vektorių sekos skaičiavimas, kurie kaupia savyje informaciją apie duotą garso įrašo signalą. Šis procesas dažniausiai aprašomas 3

etapais. Pirmasis vadinamas kalbos analize, kuris išskiria dažnių spektrą ir sugeneruoja dar neapdorotus požymius, apibūdinančius trumpų kalbos intervalų galios spektrą. Antrojo etapo metu yra sukompiliuojami išplėstiniai požymių vektoriai, sudaryti iš statinių ir dinaminių požymių. Galiausiai šie išplėstiniai vektoriai transformuojami į glaustus, bei informatyvius vektorius, kurie perduodami atpažintuvui. [7]

Yra keletas skirtingų požymių išskyrimo metodų: *suvokiminė tiesinė prognozė (*angl. *Perceptual Linear Prediction), tiesinės prognozės kodavimas (*angl. *Linear Predictive Codes)*, bet dažniausiai naudojami yra *Melų dažnių skalės kepstriniai koeficientai..*

1.2.2. Melų dažnių skalės kepstriniai koeficientai (MFCC)

MFCC yra reprezentacija realaus kepstro, kuris gaunamas pritaikius *Fourje transformaciją* duotam skaitmeniniui kalbos signalui. *MFCC* skiriasi nuo realaus kepstro, nes šiems koeficientams yra pritaikyta netiesinė dažnių skalė, kuri atspindi žmogaus klausos sistemos elgseną. Be to , šie koeficientai yra patikimi ir pritaikomi įvairiems kalbėtojams, atsižvelgiant į aplinkinį triukšmą. Kalbos signalas visų pirma išskaidomas į atskirus trumpo laiko intervalo kadrus. Kiekvienas kadras yra apdorojamas taip, kad signalas būtų tolydinis, panaudojant *Hamming'o* filtrą. Filtro koeficientai w(n) apskaičiuojami pagal formulę:

$$w(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi}{N-1}\right), 0 \le n \le N-1, 0\ kitu\ atveju; \quad (1)$$

čia N – visas kadrų skaičius, n – duotasis kadras.

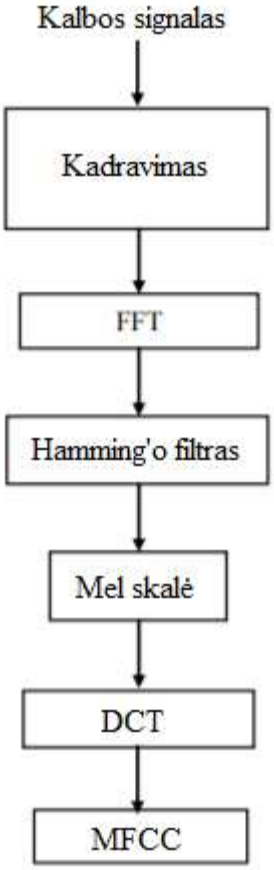
Pritaikius *Hamming'o* filtrą visiems kadrams, atliekama *greitoji Furjė transformacija* (*FFT*) kiekvienam kadrui tam, kad išskirti dažninius signalo komponentus. Toliau pritaikomas logaritminis *Mel skalės* filtras transformuotam kadrui. Ši skalė yra tiesinė dažniams iki 1kHz, ir logaritminė aukštesnio lygio dažniams. Priklausomybė tarp kalbos dažnio ir *Mel skalės* išreiškiama formule:

$$f = 2595\log(1 + f(Hz)/700; \quad (2)$$

Galiausiai, pritaikius Mel filtrą, apskaičiuojama *diskretinė kosinuso transformacija* (*DCT*), kuri išskiria reikšmingiausius kalbos parametrų koeficientus.

Kiekvienam kadrui yra apskaičiuojamas *MFCC* koeficientų rinkinys – akustinis vektorius, kuris savyje turi fonetiškai svarbias kalbos charakteristikas. Apskaičiuoti akustiniai vektoriai naudojami kituose kalbos atpažinimo kūrimo etapuose.

Bendrą algoritmo vaizdą galima pamatyti 3 pav.



3 pav. MFCC išvestis.

1.2.3. Dekodavimas

Dekodavimas laikomas svarbiausiu žingsniu kalbos atpažinimo procese. Dekodavimo funkcija skirta geriausio atitikmens suradimui duotiems požymių vektoriams, panaudojant turimą duomenų bazę. Dekoderis atlieka faktinį sprendimą apie duotos iteracijos atpažinimą, kombinuojant ir optimizuojant informaciją gautą iš akustinio ir kalbos modelio.

1.2.4. Akustinis modeliavimas

Pagrindinis akustinio modeliavimo tikslas yra pagerinti kalbos atpažintuvo tikslumą, nurodant atpažinimo vienetus ir parinkti labiausiai tinkančius akustinius požymius fonemai, kurią reikia atpažinti. K. Driaunio disertacijoje [8] rašoma:

> *Realizuojant automatinio kalbos atpažinimo sistemas pirmas ir vienas iš svarbiausių sprendimų – pasirinkti atpažinimo vienetus. Šie vienetai gali būti skirtingo ilgio (trukmės) ir kiekvienas pasirinkimas turi tiek savus privalumus, tiek trūkumus ir dažniausiai yra sąlygojamas atpažinimo sistemai keliamų tikslų. Akustinis*

modeliavimas – tai procesas, kurio metu yra generuojami analizuojamų kalbos vienetų (žodžių, morfemų, fonemų ar kt.) akustiniai modeliai. Egzistuoja platus akustinių modelių bei modeliavimo technologijų spektras.

Akustinis modeliavimas gali būti apibrėžtas kaip kalbos signalo požymių vektoriaus statistinės reprezentacijos apskaičiavimas. Yra keletas metodų, naudojamų akustinio modelio sukūrimui – *paslėptas Markovo modelis*, *sąlyginiai atsitiktiniai laukai* (angl. *conditional random fields*), *segmentiniai modeliai*, *gilieji neuroniniai tinklai* ir kt.

1.2.5. Kalbos modelis

Kalbos modelis papildo automatinio kalbos atpažinimo sistemą kalboje esančia lingvistine informacija. Dažnai naudojami statistiniai kalbos modeliai, atsižvelgiant į tai, kad šnekamojoje kalboje vartojama sakinių struktūra dažniausiai neatitinka kalbos taisyklių.

Statistinio kalbos modelio paskirtis yra kaupti tikimybinius rodiklius apie įvairias žodžių kombinacijas. Remiantis K. Driaunio disertacija [8], labiausiai paplitęs tikimybinis metodas yra *n-gram*, kuris modeliuoja šalia einančių n žodžių (1 – unigramų, 2 – bigramų, 3 – trigramų) pasirodymo tekste tikimybes. Taip pat visaverčiam kalbos modeliavimui yra naudojami tokie kalbos požymiai, kaip veiksmažodžių asmenavimo, daiktavardžių linksniavimo taisyklės, kadangi šie efektai žymiai apsunkina atpažinimą akustiniu lygmeniu – tas pats žodis akustiškai skamba panašiai, tačiau bendram sakinio kontekste jo reikšmė gali būti pakitusi.

Be plačiai naudojamo statistinio kalbos modelio galima naudoti [6]:

1. *unikalų modelį* – kiekvienas žodis turi vienodą pasirodymo tekste tikimybę;
2. *stochastinį modelį* – žodžio pasirodymo tekste tikimybė priklauso nuo prieš tai esančio žodžio;
3. *ribotos būsenos modelį* (angl. *finite state*) – žodžių sekai nuspėti naudojamas ribotos būsenos tinklas;
4. *laisvo konteksto gramatikos modelis* – naudojamas konkretiems sakiniams koduoti.

1.3. Akustinio modeliavimo metodai

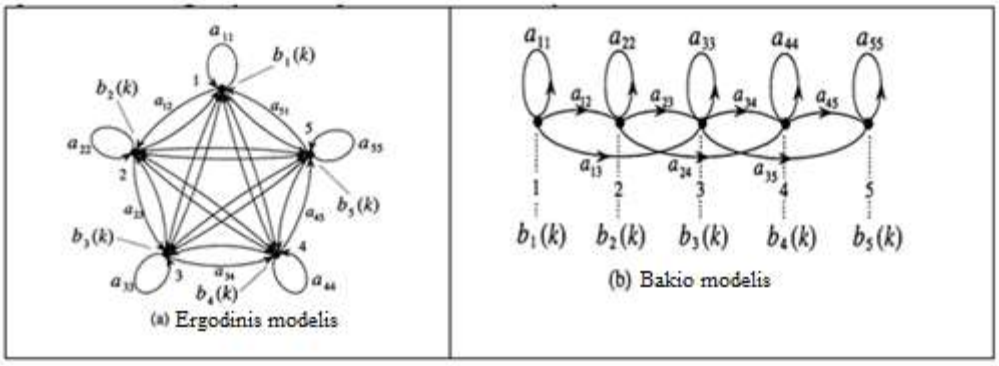
1.3.1. Paslėptas Markovo modelis

Svarbus terminas kalbant apie *paslėptą Markovo modelį* yra Markovo grandinės – jos modeliuojamos taip tarsi kiekvienas įvykis yra sujungtas su prieš tai esančiu įvykiu su atitinkama tikimybe iš kiekvieno įvykio pereiti į kitą.

Išskiriamos dvi Markovo grandinių struktūros:

1. *ergodinis* arba pilnai sujungtas modelis;
2. *iš kairės į dešinę* modelis arba *Bakio* modelis.

Modeliuojant pirmąjį modelį, struktūra yra apibrėžiama taip, kad iš bet kurios būsenos į kitą galima pereiti per vieną žingsnį. Antroji struktūra – galimos trys variacijos patekimui iš vienos būsenos į kitą: galima judėti į sekančia, dar sekančia arba pasilikti toje pačioje būsenoje. Toliau yra pateikiamas šių dviejų modelių grafinis vaizdas 4 pav.



4 pav. Markovo grandinių struktūros. (a) ergodinis modelis, (b) Bakio modelis. [9]

Būtent automatinio kalbos atpažinimo sistemai kurti pasirenkama antroji Markovo modelio struktūra, kadangi ji atspindi šnekamojoje kalboje vykstančius procesus: fonetiniai procesai vyksta vienas po kito, o kai kurie fonetiniai įvykiai kartais gali būti praleidžiami dėl garsų asimiliacijos arba tarimo klaidų [9].

Doktorantų knygoje [9] teigiama, kad sėkmingai panaudoti paslėptą Markovo modelį kalbos signalų atpažinimui reikia išspręsti tris uždavinius:

1. įvertinimo uždavinys: turint stebėjimų seką O={O1,O2,…,OT} ir grandinę aprašančio modelio parametrus λ={A,B,π} reikia apskaičiuoti tikimybę P(O|λ), kad duota stebėjimų seka buvo sugeneruota duoto modelio;
2. paslėptų būsenų nustatymo uždavinys: turint stebėjimų seką O={O1,O2,…,OT} reikia nustatyti būsenų seką I={i1,i2,…,iT}, kuri būtų optimali tam tikro pasirinkto prasmingo kriterijaus prasme;
3. apmokymo uždavinys: kaip parinkti modelio parametrus λ={A,B,π}, kad būtų maksimizuota tikimybė P(O|λm).

1.3.2. Monofoninis metodas

Monofoninis metodas yra grįstas prieš tai aprašytu *paslėptu Markovo modeliu*. Žodžio tarimas pateikiamas kaip serija simbolių, kurie atitinka atskirus garso vienetus. Tokie simboliai vadinami fonemomis.

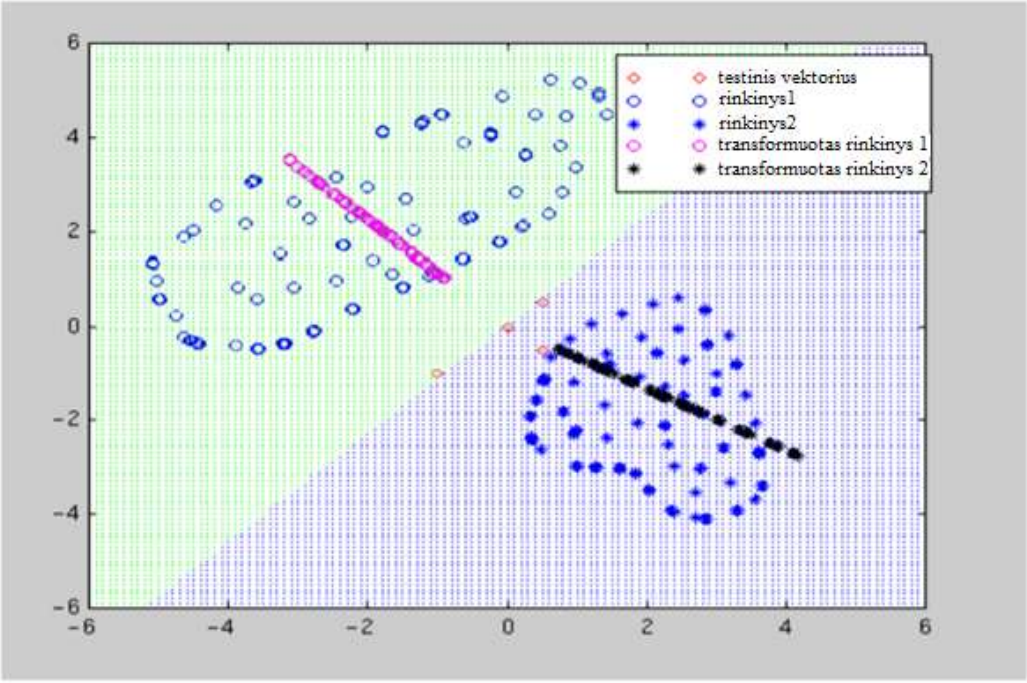
1.3.3. Trifoninis metodas

Taip pat grįstas p*aslėptu Markovo modeliu*. *Trifoninis* metodas – 3 fonemų grupė, kurios forma yra „K-X+D“ – čia K: fonema esanti iš kairės X fonemos atžvilgiu, X: duotoji fonema, D: fonema esanti iš dešinės X fonemos atžvilgiu.

1.3.4. Linijinė diskriminanto analizė (LDA)

Linijinė diskriminanto analizė skirta turimų duomenų klasifikavimui ir dimensijų sumažinimui. Šis metodas pritaikomas jau apdorotiems *MFCC* kadrams ir dar labiau išskiria duomenų požymius. *LDA* pagrindinis tikslas yra surasti duotų požymių vektorių *X*, *n* dimensijų erdvėje linijinę transformaciją į požymių vektorius *Y*, *m* dimensijų erdvėje, kur $m < n$ su maksimaliu duomenų klasifikavimu. [10]

Straipsnyje [11] yra aprašyti visi žingsniai algoritmui pritaikyti, ir vaizdinė duomenų klasifikavimo forma pateikiama 5 pav.:

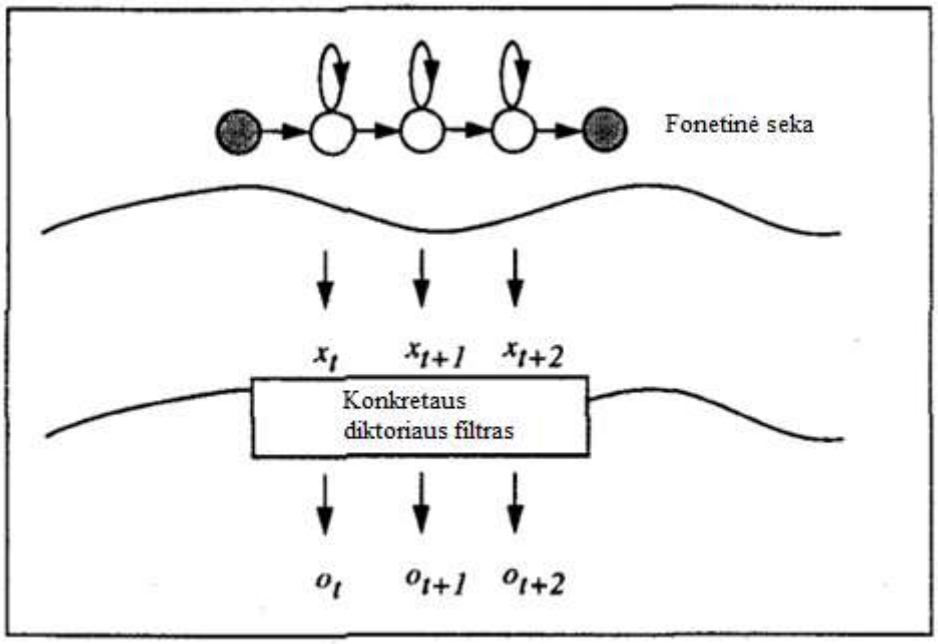


5 pav. Duomenų rinkiniai, atvaizduoti originalioje erdvėje ir transformuotoje erdvėje. [11]

Kartu su *LDA* metodu naudojama *didžiausios tikimybės linijinė transformacija* (angl. *Maximum Likelihood Linear transform*). Čia atliekama papildoma transformacija panaudojant linijinę matricą dar prieš *paslėto Markovo modelio* grandinių būsenų apmokymą [12].

1.3.5. Prisitaikantis diktoriaus mokymas (SAT)

SAT formuluotė pagrįsta generuojančio kalbos proceso pagrindu, kurį sudaro dvi komponentės. Viena iš jų atvaizduoja fonetiškai svarbių kalbos įvykių variacijas, kurios nepriklauso nuo konkretaus kalbėtojo. Šios variacijos realizuojamos panaudojus *HMM* grįstų akustinių modelių rinkinį. Paveiksle 6, 3 būsenų *HMM* generuoja fonetinę seką x_{t1}, x_{t+1}, x_{t+2}, kuri yra nepriklausoma nuo konkretaus kalbėtojo. Antrasis komponentas savyje turi informaciją apie konkretaus kalbėtojo kalbos akcentą, regioninį dialektą, balso tembrą, amžių bei lytį. Šis antrasis komponentas veikia kaip filtras, kuris transformuoja fonetinius kalbos ypatumus, ištartus konkretaus kalbėtojo, taip priskirdamas tik jam išskirtus kalbėsenos atributus. Nuo kalbėtojo nepriklausanti įvykių seka yra transformuojama perleidžiant per filtrą į nuo kalbėtojo priklausančią įvykių seką 6 pav. [13]



6 pav. *SAT* modelis konkretaus diktoriaus kalbėsenos modeliavimui. [13]

1.3.6. Tarpinis Gauso mišinių modelis (SGMM)

Paprasčiausios formos modelis gali būti išreiškiamas tokiomis 3 formulėmis:

$$p(x|j) = \sum_{i=1}^{I} \omega_{ji}\, N(x;\ \mu_{ji}, \Sigma_i); \tag{3}$$

$$\mu_{ji} = M_i v_j; \tag{4}$$

$$\omega_{ji} = \frac{\exp w_i^T v_j}{\sum_{i\prime=1}^{I} \exp w_{i\prime}^T v_j} ; \quad (5)$$

kur $x \in R^D$ yra požymis, j – kalbos būsena, $v_j \in R^S$ yra būsenos vektorius su $S \cong D$, kuris yra tarpinėje dimensijoje, ir modelis kiekvienoje būsenoje yra paprastas *GMM* su I *Gausų* (angl. *Gaussians*), *mišinių svoriais* ω_{ji} (angl. *mixture weights*), *įverčių* μ_{ji} (angl. *means*) ir bendrų būsenų *kovariacijų* $\sum_i$ (angl. *covariances*). Čia *mišinių svoriai* ir *įverčiai* nėra modelio parametrai. Vietoje jų, perskaičiuotos reikšmės iš konkrečios būsenos vektoriaus $v_j \in R^S$ su tarpine dimensija S, kuri tipiškai yra beveik tokia pat kaip ir požymių dimensija D, išreikšta per globalius parametrus M_i ir w_i. Konkrečios būsenos parametrai v_j nusako *įverčius* μ_{ji} ir *svorius* ω_{ji} visiems i, kuris yra I(D+1) parametrų *Gauso* būsena, bet dimensija S tipiškai bus daug mažesnė už I(D+1), todėl modelio apimtis yra tik dalis visos parametrų erdvės. [14]

1.3.7. Neuroniniai tinklai (DNN)

Gilieji neuroniniai tinklai susideda iš iėjimo sluoksnio, kelių paslėptų sluoksnių ir išėjimo sluoksnio. Kiekvienas sluoksnis turi fiksuotą neuronų (mazgų) skaičių ir kiekviena šalia esančių sluoksnių pora yra visiškai sujungta su svorių matrica. Mazgų „*aktyvavimas*“ tam tikrame sluoksnyje apskaičiuojamas transformuojant ankstesnio sluoksnio išėjimo svorių matricą: $a^{(i)} = M^{(i)}x^{(i-1)}$. To sluoksnio išėjimas tada apskaičiuojamas pritaikant „*aktyvavimo funkciją*“ $x^{(i)} = h^{(i)}a^{(i)}$.

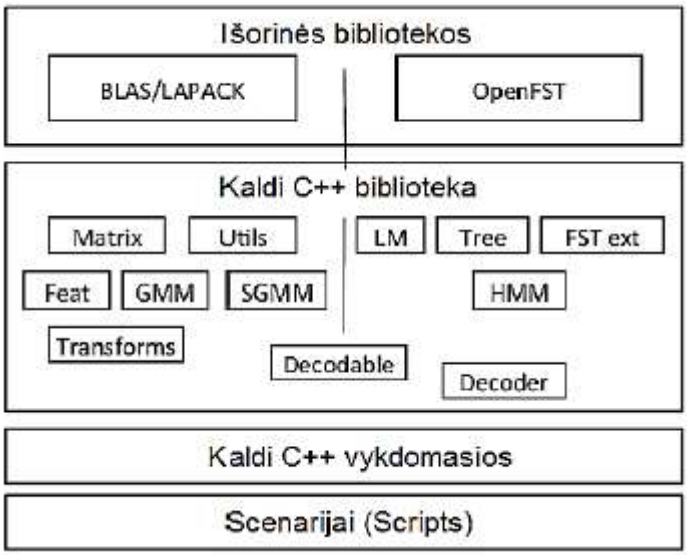
Kokio tipo aktyvavimo funkcija naudojama išėjimo sluoksnyje priklauso nuo to, kam yra naudojamas *DNN* tinklas. Jeigu *DNN* yra apmokytas regresijos būdu, išėjimo aktyvavimo funkcija yra tiesinė ir galutiniame rezultate gauname vidutinę kvadratinę paklaidą tarp išėjimo ir įėjimo. Jei *DNN* apmokytas kaip klasifikatorius, tada išėjimo aktyvavimo funkcija yra normalizuota eksponentinė funkcija (angl. *softmax)* ir rezultate gauname kryžminę entropiją tarp išėjimo ir tikrosios klasės etiketės. Klasifikatoriaus atveju, kiekvienas išėjimo neuronas yra priskirtas tam tikrai klasei ir išėjimas gaunamas kaip labiausiai tikėtinas atitikmuo įėjimo signalui. [16]

1.3.8. Neuroninio tinklo apmokymas kalbos atpažinimo sistemoje

DNN klasifikatorius gali būti panaudotas kaip akustinis modelis kalbos atpažinimo sistemoje, kuris parenka labiausiai tikėtiną subfonetinį vienetą (angl. *senone*) duotai akustinei stebėsenai. Požymių vektoriai yra išskiriami iš duoto kalbos signalo fiksuotu kadravimo dažniu panaudojant *MFCC* ar bet kurį kitą požymių išskyrimo metodą. Dekodavimui naudojamas iš anksto sugeneruotas *HMM* modelis ir *DNN* tinklas, kuris suranda labiausiai tikėtiną *senonų* seką duotam požymių vektoriui. Tam kad apmokyti neuroninį tinklą aukštu lygiu, reikia turėti iš anksto apskaičiuotus požymių išskyrimus, naudojant tokius metodus kaip *MFCC, LDA, MLLT* ir *SAT*.[17]

1.4. Kaldi programinio paketo apžvalga

Schematiškas vaizdas pateikiamas 7 pav. Svarbiausios bibliotekos yra *OpenFST* ir bibliotekos, skirtos skaitmeninei algebrai. Bibliotekos funkcijos kviečiamos panaudojant komandinę eilutę. Visi įrankiai parašyti panaudojant C++ programavimo kalbą. Galima išskirti specializuotas bibliotekas GMM (*Gaussian Mixture Models*), SGMM (*Subspace Gaussian Mixture Models*), HMM (*Hidden Markov Models*), skirta Gauso mišinių modeliams, paslėptiesiems Markovo modeliams ir pan. [18]



7 pav. Kaldi programinio įrankio struktūra ir naudojamos bibliotekos. [18]

Kaldi paketas patikimiau veikia *Linux* operacinėje sistemoje, nors galima jį vartoti ir su *Windows* operacine sistema.

1.4.1. Automatinio kalbos atpažinimo sistemos dalys Kaldi programiniame pakete

Pagrindiniai *Kaldi* paketo komponentai:

1. požymių išskyrimo komponentas: *Kaldi* leidžia rinktis tarp MFCC ir PLP požymių išlaikant jų konfigūravimo galimybes;
2. akustinio modeliavimo komponentas: jų pagrindą sudaro jau minėtos bibliotekos GMM, SGMM ir HMM. Taip pat galima naudoti kitus metodus, pvz., giliuosius neuroninius tinklus;
3. kalbos modeliavimo komponentas: šis komponentas naudojamas, kai bandoma atpažinti rišli kalba. Atpažįstant izoliuotus žodžius naudojamas supaprastintas pirmos eilės kalbos modelis;
4. dekodavimo grafų komponentas: šis komponentas reikalingas prieš testavimo procedūrą;

5. dekodavimo komponentas: tai, komponentas, atliekantis atpažinimo tikslumo testavimą. Galima rinktis paprastą dekoderį arba optimizuotą dekoderį.

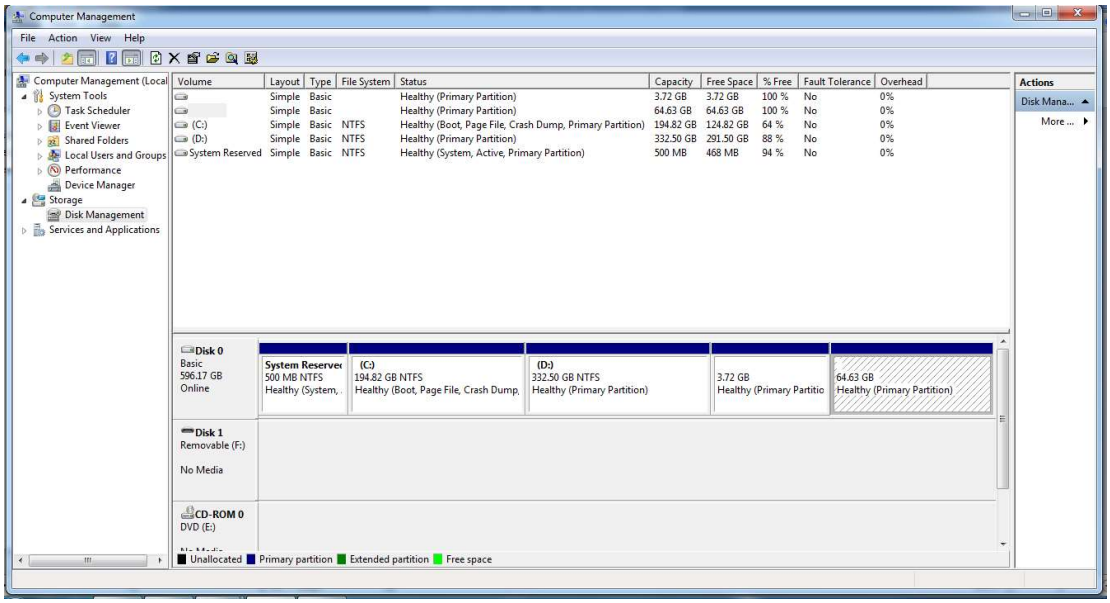
2. Metodinė dalis

2.1. Darbinės aplinkos paruošimas

2.1.1. Ubuntu sistemos įdiegimas

Ubuntu operacinės sistemos įdiegimas gali būti dvejopas: naudojant atskiras kietojo disko sritis (angl. *partitions*) – tokiu atveju esama *Windows* ar bet kokia kita sistema lieka nepaliesta ir kompiuteryje yra dvi operacinės sistemos, arba galima ištrinti jau esamą operacinę sistemą ir įrašyti *Ubuntu* operacinę sistemą kaip pagrindinę sistemą.

Šiuo atveju buvo panaudotas kietojo disko skaidymas. Tai vykdoma „*Computer Managment*“ lange pasirinkus skiltį – „*Disk Management*“. Disko būsena tokiu atveju yra suskaidoma į kelias dalis, pasirenkant, kiek norima disko atminties skirti *Ubuntu* operacinės sistemos įdiegimui.



8 pav. Disko skaidymo langas.

Kai diskas yra suskaidytas, ir dalis disko *Computer Managment* lange yra *Unallocated* – neapibrėžtas, sekant *Ubuntu* oficialaus tinklalapio [19] nurodymus pradedama diegti *Ubuntu* operacinė sistema.

2.1.2. Papildomų bibliotekų įrašymas

Kai kompiuteryje yra įdiegta *Ubuntu Linux* operacinė sistema, *Kaldi* programinio paketo įrašymui reikalingi papildomi plėtiniai operacinei sistemai:

1. *atlas* – skaičiavimų automatizavimas ir optimizavimas tiesinės algebros skaičiavimams;
2. *autoconf* – automatinis programinės įrangos rinkinys skirtingoms operacinėms sistemoms;

3. *automake* – sukurti nešiojamus *Makefile* katalogus;
4. *git* – paskirstytos peržiūros valdymo sistema;
5. *libtool* – kurti statines ir dinamines bibliotekas;
6. *svn* – peržiūros valdymo sistema, reikalinga *Kaldi* programinio paketo parsiuntimui ir įdiegimui;
7. *wget* – duomenų perdavimas naudojant *HTTP, HTTPS, FTP* protokolus;
8. *zlib* – duomenų suspaudimas.

Visi šie plėtiniai yra įdiegiami naudojant *Ubuntu* sistemos terminalą, kur yra vykdomos *Unix* tipo komandos. Naudojama komanda bibliotekoms įdiegti – *sudo apt-get*.

Sistemos plėtiniams patikrinti, *Kaldi* aplanke galima rasti kūrėjų parašytą kodą *check_dependencies.sh*. Šis kodas tikrina visus operacinės sistemos plėtinius, reikalingus *Kaldi* programiniam paketui paleisti, bei duoda instrukcijas, ką reikia daryti, norint įdiegti nesamą plėtinį. Toliau pateikiamas *check_dependencies.sh* kodo fragmentas:

```
if ! dpkg -l | grep -E 'libatlas3gf|libatlas3-base' >/dev/null; then
  echo "You should probably do: "
  echo " sudo apt-get install libatlas3-base"
  printed=true
```

9 pav. *Check_dependencies.sh* failo kodo fragmentas.

Šiuo atveju, programa tikrina, ar sistemoje yra įdiegta *atlas* biblioteka. *Echo* komandos suveikia, kai *if* sąlyga yra patenkinama, ir terminale išvedamos instrukcijos *atlas* bibliotekos įdiegimui.

Kaldi programinį paketą galima diegti, kai kompiutyje yra sukonfigūruota *Ubuntu* operacinė sistema su papildomais aukščiau pateiktais plėtiniais.

2.1.3. Kaldi programinio paketo įdiegimas

Kaldi programinis paketas parsiunčiamas iš *GitHub* repositorijos pasinaudojus *git* komanda operacinės sistemos terminale:

```
git clone https://github.com/kaldi-asr/kaldi.git kaldi --origin upstream
cd kaldi
```

10 pav. *Git* bibliotekos komanda, naudojama Kaldi programiniam paketui parsiųsti iš *GitHub* repositorijos.

Parsiuntus *Kaldi* aplanką, jame galima rasti *install.txt* failą, kuriame surašytos instrukcijos kaip įdiegti *Kaldi* programinį paketą. Įdiegimas suskirstytas dalimis:

1. Plėtinių patikrinimas.

2. Kaldi programinio paketo kompiliavimas

Pirmoji dalis aprašyta 2.1.2. skyrelyje. Įvykdžius pirmą dalį, terminale surandamas aplankas *Kaldi/src* ir įvykdomos 11 pav. pateiktos komandinės eilutės:

```
./configure --shared
make depend -j 8
make -j 8
```

11 pav. Komandinės eilutės skirtos *Kaldi* programiniam paketui sukompiliuoti.

Čia „–j 8“ reiškia, kad kompiliavimas bus vykdomas lygiagrečiai (vienu metu vykdomos 8 užduotys). Šioje vietoje reikia atsižvelgti į kompiuterio resursus, kadangi 8 užduočių vykdymas vienu metu gali būti sudėtingas seniems ar silpniems kompiuteriams.

2.2. Garsyno aprašymas

Šiame darbe buvo naudojami dviejų skirtingų dydžių garsynai – 30 ir 100 diktorių. Pirmasis garsynas buvo naudojamas kuriant paprastą akustinį modelį, panaudojant *HMM* grįstą monofoninį metodą. Bandymų metu, gauta apytiksliai vos 1% paklaida. Buvo nuspręsta didinti diktorių skaičių iki 100, bei panaudoti 5dB foninį triukšmą ir atlikti bandymus su daugiau apmokomųjų ir testavimo *.wav* failų.

2.2.1. 30 diktorių garsynas

Lietuviškų skaičių nuo 0 iki 9 garso įrašų rinkinį *SKAIC30* sudaro trisdešimties diktorių balso įrašai: 23 moterys ir 7 vyrai. Kiekvieno failo pavadinimas yra sudarytas iš lyties indikatoriaus (pirmoji raidė), pirmų trijų vardo raidžių, pirmų trijų pavardės raidžių, skaičiaus, kurį diktorius ištaria bei dviejų skaitmenų iteracijos – pvz. *FAGNGRA000*. Kiekvienas skaičius ištariamas 20 kartų. 30 diktorių, lietuviškų skaičių pavadinimų garso įrašų rinkinys sudarytas iš 6000 skirtingų balso įrašų, diskretizacijos dažnis – 16 kHz, viena atskaita koduojama 16 bitų. Apmokomiesiems duomenims priskiriami 24 diktorių garso įrašai, testavimo duomenims – 6 diktorių garso įrašai.

2.2.2. 100 diktorių garsynas

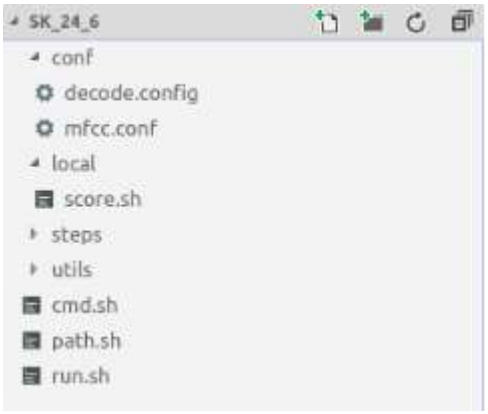
Šis garsynas sudarytas iš 100 diktorių, įskaitant prieš tai minėto 30 diktorių garsyno diktorius. Iš viso šį garsyną sudaro 62 moterys ir 38 vyrai po 20 kiekvieno skaičiaus ištarimų. Tam, kad nustatyti kaip kuris metodas atpažįsta šį garsyną, t.y., nustatyti atpažinimo dėsningumus, garsynas buvo užtriukšmintas baltuoju triukšmu 5 dB lygyje. Garsyną sudaro 20000 garso įrašų, kurių diskretizavimo dažnis yra 16 kHz, o atskaitos bitų skaičius – 16. Apmokymui atrinkti 80 diktorių

garso įrašai, o testavimui – likusiųjų 20 diktorių garso įrašai. Numatytas garsyno atpažinimo kryžminis patikrinimas keičiant apmokymo ir testavimo garso įrašus.

2.3. Automatinio kalbos atpažintuvo kūrimas su Kaldi programiniu paketu

2.3.1. Projekto sukūrimas

Norint sukurti savo projektą reikia sukurti aplanką *kaldi/egs* direktorijoje. Šiuo atveju sukuriamas aplankas pavadinimu *sk_24_6*. Kad būtų paprasčiau rašyti pagrindinį *run.sh* skriptą, rekomenduojama įsikelti pagrindinius *Kaldi* įrankius utils ir steps į *sk_24_6* katalogą iš *Kaldi* pakete esančio projekto *kaldi/egs/wsj/s5*. Rezultatų dekodavimui reikalingas dekodavimo skriptas *score.sh*, kurį galima rasti *kaldi/egs/voxforge/local*. Reikalingi konfigūraciniai projekto failai kopijuojami iš *kaldi/voxforge/conf* katalogo. Projekto kūrimo pradžioje galima iškart sukurti konfigūracinius *cmd.sh*, *path.sh* failus, bei pagrindinį *run.sh* skriptą, kuris bus aprašytas 2.3.5 skyrelyje. Sukurto aplanko struktūra pavaizduojama 12 pav.



12 pav. *Visual studio code* editoriaus langas su atidarytu *sk_24_6* projektu.

2.3.2. Garsyno paruošimas projektui

Ruošiant bet kurį garsyną darbui su Kaldi paketu reikia paruošti penkis pagalbinius failus, susiejančius diktorius su garso įrašais.

Informacija apie diktoriaus lytį, failas spk2gender. Failo *spk2gender* struktūra turi būti *diktoriaus_indentifikatorius lyties_indikatorius*. Sekančiame paveikslėlyje parodytas šio failo fragmentas:

spk2gender

```
1   FAGNGRA f
2   FAGNVIN f
3   FAISIZI f
4   FAISZYM f
5   FAUSNEM f
6   FDAILOI f
```

13 pav. *Spk2gender* failo fragmentas.

Kalbėtojo identifikavimui pasirinkta tokia sistema: pirma raidė reiškia kalbėtojo lytį (f-female, m-male), sekančios 3 raidės reiškia vardą, o po to sekančios trys raidės – pavardę

Failas utt2spk – sąsaja tarp garso įrašo ir kalbėtojo. Failas sudarytas iš garso įrašo ir kalbėtojo identifikatorių. Paveikslėlyje pateikiamas šio failo fragmentas:

utt2spk

```
1   FAGNGRA000 FAGNGRA
2   FAGNGRA001 FAGNGRA
3   FAGNGRA002 FAGNGRA
4   FAGNGRA003 FAGNGRA
5   FAGNGRA004 FAGNGRA
6   FAGNGRA005 FAGNGRA
7   FAGNGRA006 FAGNGRA
8   FAGNGRA007 FAGNGRA
9   FAGNGRA008 FAGNGRA
10  FAGNGRA009 FAGNGRA
11  FAGNGRA010 FAGNGRA
12  FAGNGRA011 FAGNGRA
13  FAGNGRA012 FAGNGRA
14  FAGNGRA013 FAGNGRA
```

14 pav. *Utt2spk* failo turinio fragmentas.

Garso įrašą nusako kalbėtojo identifikatorius kartu su skaičiaus pavadinimu nuo 0 iki 9 ir ištarimo numeriu nuo 00 iki 19.

Failas text – sąsaja tarp garso įrašo ir įrašo transkripcijos. Failas sudarytas iš garso įrašo identifikatoriaus ir įrašo transkripcijos. Paveikslėlyje pateikiamas šio failo fragmentas:

text.txt

```
FAGNGRA000 nulis
FAGNGRA001 nulis
FAGNGRA002 nulis
FAGNGRA003 nulis
FAGNGRA004 nulis
FAGNGRA005 nulis
FAGNGRA006 nulis
FAGNGRA007 nulis
FAGNGRA008 nulis
FAGNGRA009 nulis
FAGNGRA010 nulis
FAGNGRA011 nulis
FAGNGRA012 nulis
FAGNGRA013 nulis
```

15 pav. *Text* failo turinio fragmentas.

Failas wav.scp – sąsaja tarp garso įrašo ir identifikatoriaus ir realaus garso įrašo kompiuteryje. Failas sudarytas iš garso įrašo identifikatoriaus ir garso įrašo adreso kompiuteryje. Paveikslėlyje pateikto šio failo fragmentas:

wav.scp

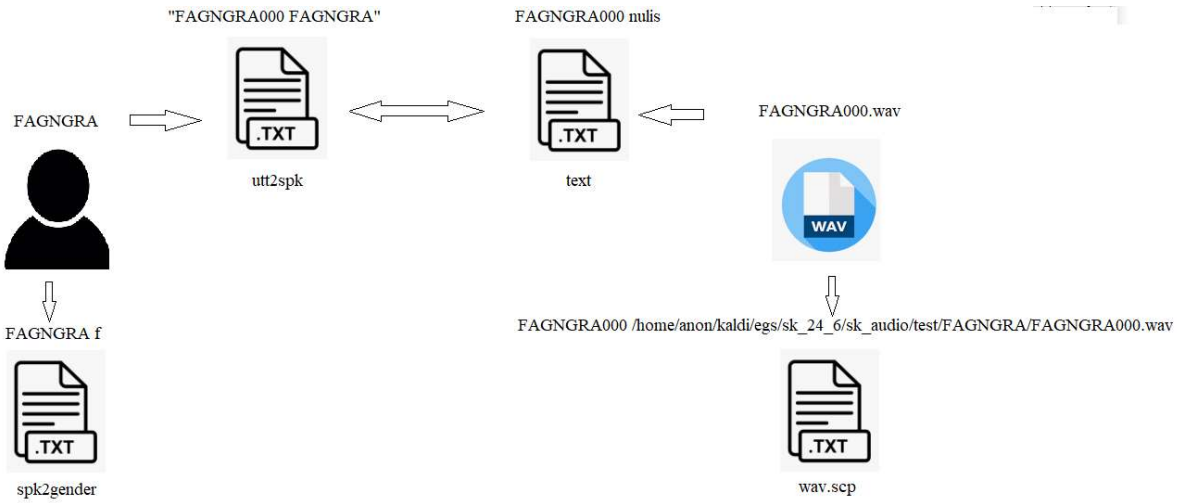
```
FAGNGRA000 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA000.wav
FAGNGRA001 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA001.wav
FAGNGRA002 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA002.wav
FAGNGRA003 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA003.wav
FAGNGRA004 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA004.wav
FAGNGRA005 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA005.wav
FAGNGRA006 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA006.wav
FAGNGRA007 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA007.wav
FAGNGRA008 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA008.wav
FAGNGRA009 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA009.wav
FAGNGRA010 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA010.wav
FAGNGRA011 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA011.wav
FAGNGRA012 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA012.wav
FAGNGRA013 /home/anon/kaldi/egs/sk_24_6/sk_audio/test/FAGNGRA/FAGNGRA013.wav
```

16 pav. *Wav.scp* failo turinio fragmentas.

Failas corpus.txt – apmokymo ir testavimo įrašų transkripcijos. Faile nurodytos visų garsinių įrašų transkripcijos – 100 diktorių garsynui yra 20000 tekstinių eilučių.

Failus *spk2gender, utt2spk, text, wav.scp* reikia sukurti testavimo ir apmokymo dalims atskirai. Failų turiniai priklauso nuo to, kokie diktoriai yra priskirti šioms dvejoms sritims. Sukurti testavimo garsyną apibūdinantys failai talpinami projekto *sk_24_6* kataloge *data/test*, apmokymų garsyną apibūdinantys failai talpinami *data/train* kataloge.

Bendras sąsajos vaizdas tarp pirmų keturių failų pateikiamas 17 pav.:



17 pav. Garsyno informacinių failų tarpusavio sąsaja.

Kaldi duomenų paruošimo failams (*spk2gender , utt2spk , text , wav.scp, corpus.txt*) sukurti buvo pasinaudota *Java* programavimo kalba. *Java* - objektiškai orientuota, bendros paskirties aukšto lygio programavimo kalba, kuria galima programuoti bet kokioje operacinėje aplinkoje, kadangi *Java* „baitkodas“ vykdomas *JVM* (Java virtualioje mašinoje). Kiekvienai operacinei sistemai yra sukurta *Java virtuali mašina*, kurią reikia atsiųsti iš oficialaus tinklalapio [20]. Java „baitkodas“ vykdymo metu yra „*išverčiamas*“ į procesoriui suprantamą kalbą.

Prieduose yra pateiktas *Java* kalbos kodas. Metode *public static void main(String[] args)* kviečiama pagrindinė *Kaldi* klasės funkcija *public static void makeTxtFileWAV(File wavFolder, File newFile, string txtType)*, kuri veikia rekursiniu būdu. Esminis veikimo principas yra:

1. nurodytame kataloge paimamas visų jame esančių failų sąrašas;
2. failų sąrašas yra iteruojamas, ir tikrinama ar tas failas yra aplankas;
3. jei failas yra aplankas, funkcija *makeTxtFileWAV* kviečia pati save ir kartojasi 1, 2 punktai;
4. suradus failą, su plėtiniu .wav, visas direktorijos kelias yra įrašomas į atskirą galutinių direktorijų sąrašą wavFiles;
5. priklausomai nuo to, koks *txtType* parametras paduodamas į pagrindinę funkciją *public static void makeTxtFileWAV*, failo kelias yra pakoreguojamas taip, kad atitiktų *Kaldi* reikalavimus;
6. rekursiniu būdu surinkus visus *.wav* kelių failų duomenis į sąrašą *wavFiles*, jis yra išrikiuojamas ir pasinaudojus Java bibliotekos klase *java.io.FileWriter* sukuriami reikiami txt failai.

Taip pat yra sukurta funkcija *generateCorpus()*, kuri sugeneruoja *corpus.txt* failą.

Šių failų generavimas su aukšto lygio programavimo kalba paspartina tyrimų eigą, eliminuoja žmogiškojo faktoriaus klaidas.

2.3.3. Fonemų aprašomieji failai

Fonemų aprašomuosius failus sudaro: *lexicon.txt, nonsilence_phones.txt, silence_phones.txt, optional_silence.txt.*

Lexicon.txt. Tyrimams, kaip fonetinės transkripcijos panaudotos grafemos, t.y. mažiausi rašto kalbos vienetai, atitinkantys garsinės kalbos vienetus – fonemas, išskyrus tai, kad dvibalsiai „ie“ ir „uo“ sudaro atskiras grafemas.

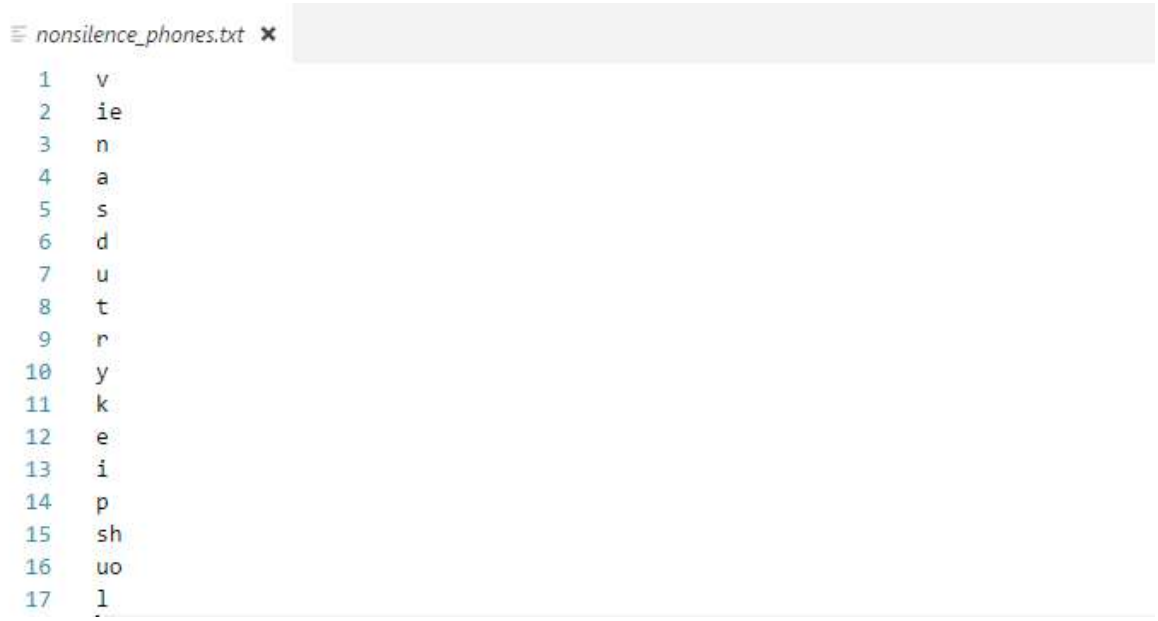
lexicon.txt

```
1   !SIL sil
2   <UNK> spn
3   nulis n u l i s
4   vienas v ie n a s
5   du d u
6   trys t r y s
7   keturi k e t u r i
8   penki p e n k i
9   sheshi sh e sh i
10  septyni s e p t y n i
11  ashtuoni a sh t uo n i
12  devyni d e v y n i
```

18 pav. *Lexicon.txt* failo turinys.

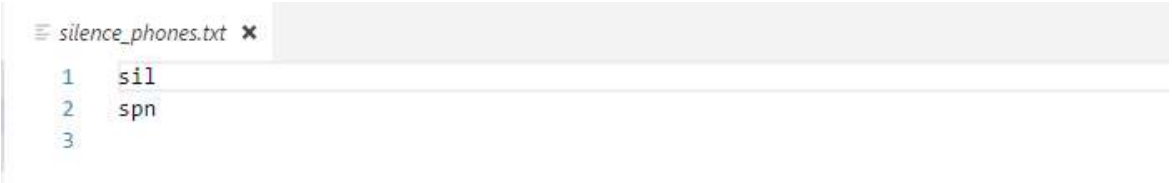
Pirmame tyrimų etape kaip fonetinės transkripcijos panaudotos grafemos. Tolimesniuose tyrimuose naudojamos *SAMPA-LT* fonetinės transkripcijos.

Nonsilence_phone.txt. Fonemų sąrašo (be tylos fonemos) failas:



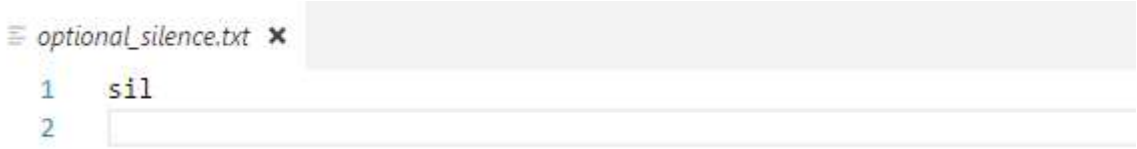
19 pav. *Nonsilence_phones.txt.* Tariamų fonemų (grafemų) sąrašas.

Silence_phones.txt. Tylos fonemų sąrašo failas:



20 pav. *Silence_phone.txt*. Tylos fonemų sąrašas.

Optional_silence.txt. Papildomų tylos fonemų sąrašo failas:



21 pav. *Optional_silence.txt*. Papildomų tylos fonemų sąrašas.

Aprašomieji fonemų failai talpinami projekto *sk_24_6* aplanke *data/local/dict*.

2.3.4. Papildomi konfigūraciniai failai

Sukurtam projektui reikalingi papildomi konfigūraciniai failai, kurie talpinami *kaldi/egs/sk_24_6/conf* kataloge:

Mfcc.conf. Šiame faile nustatomi *MFCC* metodo požymių išskyrimo proceso parametrai (pvz. kadravimo dažnis):

mfcc.conf

```
1  --use-energy=false
2  --sample-frequency=20000
```

22 pav. *Mfcc.conf* konfigūracinio failo turinys.

***Decode.config*.** Dekodavimo operacijos parametrai:

decode.config

```
1  first_beam=10.0
2  beam=13.0
3  lattice_beam=6.0
```

23 pav. *Decode.config* konfigūracinio failo turinys.

Taip pat reikalingas papildomas *path.sh* failas, kuris aprašo pagrindinę Kaldi programos direktoriją, garsyno failų direktoriją ir pan.:

path.sh

```
1  # Defining Kaldi root directory
2  export kaldi=`pwd`/../..
3
4
5  # Defining audio data directory (modify it for your installation directory!)
6  export data="/home/anon/kaldi/egs/sk_24_6/sk_audio"
7
8  # Setting paths to useful tools
9  export PATH=$PWD/utils/:$kaldi/src/bin:$kaldi/tools/openfst/bin:$kaldi/src/fstbin/:$kaldi/src/gmmbin/:$kaldi/src/gmmbin/
10
11 # Enable SRILM
12 source $kaldi/tools/env.sh
13
14 # Variable needed for proper data sorting
15 export LC_ALL=C
```

24 pav. *Path.sh* konfigūracinio failo turinys.

Galiausiai reikalingas *cmd.sh* failas, kuris aprašo, kokie kompiuterio resursai bus naudojami programai, pvz. ar naudojamas lokalus tinklo kompiuteris, ar didesnės apimties projektams naudojamas galingas serveris. Lokalaus kompiuterio konfigūraciniai parametrai pateikiami 24 pav.:

cmd.sh

```
1  #using local processor
2  export train_cmd=run.pl
3  export decode_cmd=run.pl
4
```

25 pav. *Cmd.sh* konfigūracinio failo turinys.

Path.sh ir *cmd.sh* failai talpinami pagrindiniame *kaldi/egs/sk_24_6* projekto kataloge.

2.3.5. Pagrindinis run.sh skripto aprašymas

Šiame faile pateikiama informacija apie visas kviečiamas funkcijas ir metodus. Tai gali būti duomenų nuskaitymas, jų perrikiavimas, MFCC požymių skaičiavimas, požymių normalizavimas, kalbos modelio sudarymas, akustinis modeliavimas pasirinktu metodu bei atpažinimo klaidos skaičiavimas iškviečiant dekodavimo ir testavimo metodus.

Atpažinimas naudojant monofoninį metodą. Pradiniame etape kviečiami konfigūraciniai failai (26 pav.). Juose nurodomos naudojamos bibliotekos, garso įrašų saugojimo katalogas ir pan.:

```
8
9    . ./path.sh || exit 1
10   . ./cmd.sh || exit 1
11
```

26 pav. Konfigūracinių nustatymų skriptas

Skriptas *utt2spk_to_spk2utt.pl* sukuria dar vieną papildomus failus:

```
51   # Making spk2utt files
52   utils/utt2spk_to_spk2utt.pl data/train/utt2spk > data/train/spk2utt
53   utils/utt2spk_to_spk2utt.pl data/test/utt2spk > data/test/spk2utt
```

27 pav. Failų paruošimo skriptas

Sukuriami papildomi failai *spk2utt.txt* apmokymui ir testavimui.

Skriptas *make_mfcc.sh* iš katalogo *steps* skaičiuoja MFCC požymius atskirai apmokymui ir testavimui:

```
64   steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/train exp/make_mfcc/train $mfccdir
65   steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/test exp/make_mfcc/test $mfccdir
```

28 pav. Požymių išskyrimo skriptas

Šiame skripte esantis parametras *--nj* nurodo multiprocesorinio darbo galimybę. Pasirinkta reikšmė *nj* = 1, kadangi tyrimai atliekami atskirame kompiuteryje, o ne serveryje. Toliau nurodoma iš kur paimti apmokymui skirtus parametrus, iš kur paimti garso įrašus ir kur dėti suskaičiuotus MFCC koeficientus.

SRILM biblioteka naudojama kalbos modelio kūrimui. Kalbos modelio kūrimo fragmentas [21]:

```
loc=`which ngram-count`;
if [ -z $loc ]; then
if uname -a | grep 64 >/dev/null; then
        sdir=$kaldi/tools/srilm/bin/i686-m64
else
        sdir=$kaldi/tools/srilm/bin/i686
    fi
    if [ -f $sdir/ngram-count ]; then
        echo "Using SRILM language modelling tool from $sdir"
        export PATH=$PATH:$sdir
    else
        echo "SRILM toolkit is probably not installed.
            Instructions: tools/install_srilm.sh"
        exit 1
    fi
fi

local=data/local
mkdir $local/tmp
ngram-count -order $lm_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -text $local/corpus.txt -lm $local/tmp/lm.arpa
```

29 pav. Kalbos modelio skriptas

Po požymių išskyrimo ir normalizavimo vykdomas akustinių modelių apmokymas pasirinktu metodu. Apmokymas pradedamas nuo paprasčiausio monofoninio metodo tam panaudojant *train_mono.sh* skriptą. Po to kuriami dekodavimui reikalingi grafai:

```
steps/train_mono.sh  --nj "$train_nj" --cmd "$train_cmd" data/train data/lang exp/mono
utils/mkgraph.sh data/lang exp/mono exp/mono/graph
```

30 pav. Monofoninio modelio skriptas

Iš skripto matome, kad duomenys apmokymui bus imami iš *data/train*, o sukurti akustiniai modeliai bus dedami į exp/mono katalogą:

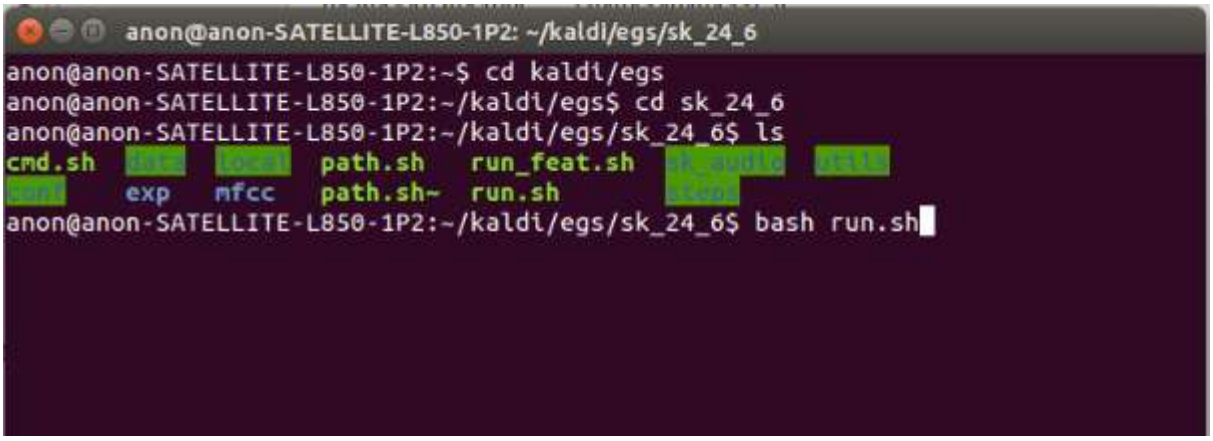
decode.sh skriptas iš *steps* katalogo atlieka testavimo procedūrą:

```
steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
 exp/mono/graph data/test exp/mono/decode
```

31 pav. Monofoninio modelio testavimo skriptas

Iš skripto matome, kad testavimo duomenys imami iš *data/test* katalogo, o testavimo rezultatai talpinami kataloge sukurto *exp/mono/decode*.

Paruoštas *run.sh* skriptas paleidžiamas Linux aplinkoje panaudojant komandą *bash run.sh,* o visas skaičiavimo procesas stebimas terminalo lange:



32 pav. *Linux* terminalo langas.

Atpažinimas naudojant trifoninį metodą. Visų sekančių atpažinimo metodų panaudojimo metodika labai panaši į atpažinimą naudojant monofoninį metodą, todėl toliau pateikiami tik akustinių modelių apmokymo skriptai, kadangi jie šiek tiek skiriasi.

Atpažinimas naudojant trifoninį metodą remiasi rezultatais, gautais iš monofoninio metodo. Papildomai atliekama lygiavimo procedūra, paruošianti duomenis sekančiam apmokymo etapui:

```
136  steps/align_si.sh --nj "$train_nj" --cmd "$train_cmd" \
137   data/train data/lang exp/mono exp/mono_ali
```

33 pav. Monofoninio metodo paruošimo trifoniniam metodui skriptas

Matome, kad apmokymo failų katalogas yra *data/train*, kalbos modelio katalogas *data/lang*, monofininio apmokymo rezultatų katalogas yra *exp/mono*, o gauti rezultatai po lygiavimo procedūros bus talpinami kataloge *exp/mono_ali*.

Trifoninis apmokymas atliekamas su *train_deltas.sh* skriptu:

```
140  steps/train_deltas.sh --cmd "$train_cmd" \
141   2000 11000 data/train data/lang exp/mono_ali exp/tri1
142  utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph
```

34 pav. Trifoninio modelio skriptas

Šiame skripte naudojami parametrai *num-leaves* = 2000 ir *tot-gauss* = 10000, nusakantys *GMM* modelio parametrus. Apmokymo failų katalogas yra *data/train*, kalbos modelio katalogas *data/lang*, informacija iš monofoninio metodo - *exp/mono_ali*, o trifoninio apmokymo rezultatai yra kataloge *exp/tri1*.

decode.sh skriptas iš *steps* katalogo atlieka testavimo procedūrą:

```
145  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
146    exp/tri1/graph data/test exp/tri1/decode
```

35 pav. Trifoninio modelio testavimo skriptas

Atpažinimas naudojant LDA+MLLT metodą. Atpažinimas naudojant LDA+MLLT metodą remiasi rezultatais, gautais iš trifoninio metodo. Papildomai atliekama lygiavimo procedūra, paruošianti duomenis apmokymo etapui:

```
152  steps/align_si.sh --nj "$train_nj" --cmd "$train_cmd" \
153    data/train data/lang exp/tri1 exp/tri1_ali
```

36 pav. LDA+MLLT modelio paruošimo skriptas

Matome, kad apmokymo failų katalogas yra *data/train*, kalbos modelio katalogas *data/lang*, trifoninio apmokymo rezultatų katalogas yra *exp/tri1,* o gauti rezultatai po lygiavimo procedūros bus talpinami kataloge *exp/tri1_ali*.

train_lda_mllt.sh skriptas naudojamas apmokymui LDA+MLLT metodu:

```
155  steps/train_lda_mllt.sh --cmd "$train_cmd" \
156    --splice-opts "--left-context=3 --right-context=3" \
157    2000 11000 data/train data/lang exp/tri1_ali exp/tri2
158  utils/mkgraph.sh data/lang exp/tri2 exp/tri2/graph
```

37 pav. LDA+MLLT modelio skriptas

Naujas parametras *splice-opts* nusako MFCC kadrų pasiskirstymą. Apmokymo failų katalogas yra *data/train*, kalbos modelio katalogas *data/lang*, informacija iš trifoninio metodo - *exp/tri1_ali*, o apmokymo rezultatai yra kataloge *exp/tri2*.

Testavimui kviečiamas *steps/decode* skriptas:

```
162  steps/decode.sh --nj "$decode_nj" --cmd "$decode_cmd" \
163    exp/tri2/graph data/test exp/tri2/decode
```

38 pav. LDA+MLLT modelio testavimo skriptas

Atpažinimas naudojant LDA+MLLT+SAT metodą. Panaudojant šį metodą gaunami požymiai, charakterizuojantys konkretų kalbėtoją. Tai pagerina visis atpažinimo sistemos tikslumą.

Atpažinimas naudojant LDA+MLLT metodą remiasi rezultatais, gautais iš LDA+MLLT metodo. Papildomai atliekama lygiavimo procedūra, paruošianti duomenis apmokymo etapui. Apmokymo

failų katalogas yra *data/train*, kalbos modelio katalogas *data/lang*, informacija iš LDA+MLLT metodo - *exp/tri2_ali*, o apmokymo rezultatai yra kataloge *exp/tri3*.

```
169  # Align tri2 system with train data.
170  steps/align_si.sh --nj "$train_nj" --cmd "$train_cmd" \
171   --use-graphs true data/train data/lang exp/tri2 exp/tri2_ali
172
173  # From tri2 system, train tri3 which is LDA + MLLT + SAT.
174  steps/train_sat.sh --cmd "$train_cmd" \
175   $numLeavesSAT $numGaussSAT data/train data/lang exp/tri2_ali exp/tri3
176
177  utils/mkgraph.sh data/lang exp/tri3 exp/tri3/graph
```

39 pav. LDA+MLLT+SAT modelio skriptas

decode_fmllr.sh skriptas naudojamas testavimui. Metodas *FMLLR* papildomai vykdo kalbėtojo identifikavimą. 40 paveikslėlyje pateiktas testavimo skripto fragmentas:

```
179  steps/decode_fmllr.sh --nj "$decode_nj" --cmd "$decode_cmd" \
180   exp/tri3/graph data/test exp/tri3/decode
181
182  steps/align_fmllr.sh --nj "$train_nj" --cmd "$train_cmd" \
183   data/train data/lang exp/tri3 exp/tri3_ali
```

40 pav. LDA+MLLT+SAT modelio testavimo skriptas

Atpažinimas naudojant SGMM metodą. Šiuo atveju naudojami skriptai: *train_ubm.sh* ir *train_sgmm2.sh*. Taip pat nurodomas prieš tai sukurto *LDA + MLLT+ SAT* akustinio metodo katalogas *exp/tri3_ali*:

```
203  steps/train_ubm.sh --nj "$train_nj" --cmd "$train_cmd" \
204   400 data/train data/lang exp/tri3_ali exp/ubm4
205
206  steps/train_sgmm2.sh --nj "$train_nj" --cmd "$train_cmd" 7000 9000 \
207   data/train data/lang exp/tri3_ali exp/ubm4/final.ubm exp/sgmm2_4
208
209  utils/mkgraph.sh data/lang exp/sgmm2_4 exp/sgmm2_4/graph
```

41 pav. SGMM modelio skriptas

Katalogas *exp/ubm4* naudojamas SGMM, kurie talpinami kataloge *exp/sgmm2_4* sukūrimui.

Testavimui naudojamas *decode_sgmm2.sh* skriptas:

```
steps/decode_sgmm2.sh --nj "$decode_nj" --cmd "$decode_cmd"\
 --transform-dir exp/tri3/decode exp/sgmm2_4/graph data/test \
 exp/sgmm2_4/decode
```

42 pav. SGMM modelio testavimo skriptas

Atpažinimas naudojant DNN metodą. Neuroninis tinklas mokomas panaudojant rezultatus, gautus iš LDA+MLLT+SAT metodo. Tai užtikrina geresnius atpažinimo rezultatus.

run_pnorm.sh skriptas panaudotas neuroninio modelio sukūrimui. Duomenys gaunami iš *exp/tri3_ali* katalogo. Galima pasirinkti ar bus dirbama atskirame kompiuteryje ar bus panaudojamas serveris su daug procesorių ir grafiniu procesoriumi:

```
dnn_cpu_parallel_opts=(--minibatch-size 128 --max-change 10 --num-jobs-nnet 8 --num-threads 16 \
                       --parallel-opts "--num-threads 16")
dnn_gpu_parallel_opts=(--minibatch-size 512 --max-change 40 --num-jobs-nnet 4 --num-threads 1 \
                       --parallel-opts "--gpu 1")

steps/nnet2/train_pnorm_ensemble.sh \
  --mix-up 5000  --initial-learning-rate 0.008 --final-learning-rate 0.0008\
  --num-hidden-layers 4 --pnorm-input-dim 2000 --pnorm-output-dim 200\
  --cmd "$train_cmd" \
  "${dnn_cpu_parallel_opts[@]}" \
  --ensemble-size 4 --initial-beta 0.1 --final-beta 5 \
  data/train data/lang exp/tri3_ali exp/tri4_pnorm
```

43 pav. DNN modelio skriptas

Mūsų atveju pasirinktas darbas atskirame kompiuteryje.

Apmokymo greitis, paslėptų sluoksnių skaičius, neuronų skaičius paslėptuose sluoksniuose pasirenkami nustatant papildomus parametrus, pvz., paslėptų sluoksnių skaičių nurodo parametras *num-hidden-layers 4.*

Kiti parametrai: apmokymo failų katalogas *data/train*, kalbos modelio katalogas *data/lang*, akustinio modeliavimo rezultatai po LDA+MLLT+SAT metodo taikymo kataloge *exp/tri3_ali*, neuroninio tinklo akustinių modelių katalogas *exp/tri4_pnorm.*

steps/nnet2/decode.sh skriptas naudojamas testavimui:

```
205  steps/nnet2/decode.sh --cmd "$decode_cmd" --nj "$decode_nj" "${decode_extra_opts[@]}" \
206    --transform-dir exp/tri3/decode exp/tri3/graph data/test \
207    exp/tri4_pnorm/decode | tee exp/tri4_pnorm/decode/decode.log
```

44 pav. DNN modelio testavimo skriptas

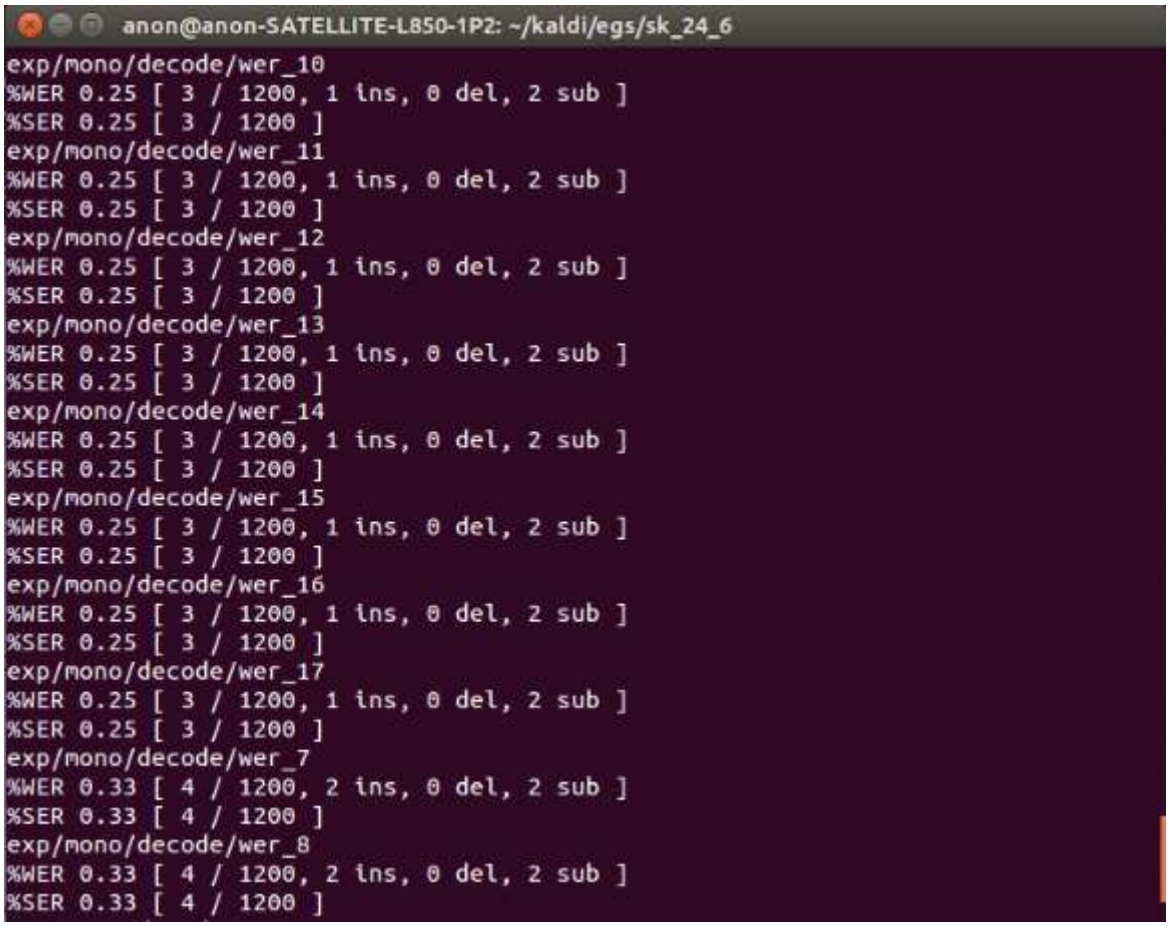
3. Rezultatai

Tyrimams panaudoti 30 ir 100 diktorių garsynai. Kiekvienam metodui atliktas penkis kartus kryžminis patikrinimas taip, kad kiekvienas diktorius pakliūtų į testavimo ir apmokymo dalis. Toliau rezultatai bus pateikiami kiekvienam metodui atskirai, ir galiausiai padarytas palyginimas, kuris automatinio kalbos atpažintuvo akustinio modeliavimo metodas suteikia tiksliausią kalbos atpažinimą.

3.1. 30 diktorių garsyno bandymų rezultatai

Tyrimų pradžioje buvo naudojamas 30 diktorių garsynas. Apmokymui skirti 24 diktorių garso įrašai talpinami *kaldi/egs/sk_24_6/sk_audio/train* direktorijoje, o testavimui skirti 6 diktorių garso įrašai *kaldi/egs/sk_24_6/sk_audio/test* direktorijoje. Tokiu principu buvo sukurti 5 atskiri garsyno aplankai, pakeičiant testavimo ir apmokymo duomenis taip, kad visi diktoriai pakliūtų į abi sritis.

Rezultatams gauti *linux* terminale paleidžiamas *run.sh* skriptas. Kiekvieno bandymo rezultatai išvedami tame pačiame terminalo lange tokiu formatu, kaip pavaizduota 45 paveiksle:



45 pav. *Linux* terminalo langas su pateiktais bandymų rezultatais.

čia matoma, kad testavimo metu buvo atliekami 10 atskirų kalbos atpažinimo bandymų, išvedant atpažinimo paklaidą procentais, padarytą klaidų skaičių, visų testavimo ištarimų skaičių, bei klaidų skaičiaus suskirstymą pagal klaidos pobūdį:

1. žodžio sumaišymas – kai vietoj ištariamo žodžio parenkamas kitas garsyno ištarimas;
2. žodžio neatpažinimas – kai automatinis kalbos atpažintuvas nepateikia jokios išvados apie ištarimą;
3. žodžio įterpimas – kai vietoj ištariamo žodžio parenkamas su garsynu jokių asociacijų neturintis ištarimas.

Toliau pateikiamos monofoninio metodo, kryžminio patikrinimo paklaidos įvertinimo rezultatų lentelės:

1 lentelė. 30 diktorių garsyno atpažinimo rezultatai. Monofoninis metodas.

Aplankas/bandymai	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
FAGNGRA FAGNVIN FAISIZI FAISZYM FAUSNEM FDAILOI	0,25	0,25	0,25	0,25	0,25	0,25	0,25	0,25	0,33	0,33
FGINGED FIEVJUR FIEVSAB FKAMMOS FLAUZET FRAISAV	0,42	0,42	0,42	0,33	0,33	0,33	0,25	0,25	0,42	0,42
FIEVVIS FJUSKIN FUGNBUC FUGNNOV FZIVSTA MLINJUR	0,92	0,75	0,75	0,75	0,67	0,67	0,67	0,50	1,17	1,00
FRUTNAN FSIMMEI FVAIVAI FVANPEC FVILVAI FVIONAB	0,67	0,42	0,42	0,42	0,33	0,25	0,25	0,17	1,25	1,08
MDAIGUS MEDGVOL MKAZANU MMODSLE MRIMAPA MVYGVAI	2,25	2,00	2,00	1,58	1,33	1,00	0,83	0,83	3,00	2,83

2 lentelė. Gautų bandymų rezultatų vidurkiai, geriausias, prasčiausias rezultatas.

	FAGNGRA FAGNVIN FAISIZI FAISZYM FAUSNEM FDAILOI	**FGINGED FIEVJUR FIEVSAB FKAMMOS FLAUZET FRAISAV**	**FIEVVIS FJUSKIN FUGNBUC FUGNNOV FZIVSTA MLINJUR**	**FRUTNAN FSIMMEI FVAIVAI FVANPEC FVILVAI FVIONAB**	**MDAIGUS MEDGVOL MKAZANU MMODSLE MRIMAPA MVYGVAI**
Vidurkis	0,266	0,359	0,785	0,526	1,765
Geriausias	0,25	0,25	0,50	0,17	0,83
Blogiausias	0,33	0,42	1,17	1,08	3,00

Iš rezultatų matyti, kad atpažinimo paklaidos diapazonas 0,359-1,765 %. Geriausias tyrimų rezultatas 0,17 %, blogiausias 3,00 %.

HTK ir *Kaldi* programinių paketų palyginimui pasinaudojama straipsnio [1] gautais rezultatais. Sistemos atpažinimo tikslumas, naudojant paprasčiausią monofoninį metodą yra panašus į atpažinimo sistemos, sukurtos su *HTK* programiniu paketu, tikslumą. Remiantis [1] straipsniu, su naudojamu tokiu pat *SKAIC30* diktorių garsynu kalbos atpažintuvas pasiekė 99 % atpažinimo tikslumą. Tuo tarpu šiame darbe sistemos, sukurtos su Kaldi programiniu paketu geriausias bandymų rezultatas siekia 99,75 % tikslumą.

3.2. 100 diktorių garsyno su foniniu 5 dB triukšmu bandymų rezultatai

Toliau pateikiami rezultatai su 100 skirtingų diktorių garsynu – 62 moterys ir 38 vyrai. Kiekvienas skaičius ištariamas 20 kartų. Šiuo atveju garso įrašų rinkinys sudarytas iš 20000 skirtingų balso įrašų. Buvo panaudoti 6 skirtingi metodai: *monofoninis*, *trifoninis*, *LDA+MLLT*, *LDA+MLLT+SAT*, *SGMM* ir *DNN*.

Rezultatams atvaizduoti išskiriami aplankų pavadinimai ir surašomi kiekvieno aplanko testavimo srityje naudojami diktoriai:

1. *A aplankas*: *FRENAPA, FRENZAP, FSALPAR, FSIMSTA, FTOMMAN, FUGNKAZ, FVAIDAN, FVAIJUO, FVESTRI, FVIRSUK, FZITKOS, FZIVOLE, MAIRVEN, MANDMAZ, MDEIJAN, MMANJIU, MMINPUT, MNERKAC, MVAIGER, MVALKAZ.*
2. *B aplankas*: *FBIRAST, FRAISAV, FRAMSTO, FRUTNAN, FSANABU, FSIMMEI, FUGNBUC, FUGNNOV, FVAIVAI, FVANPEC, FVILVAI, FVIONAB, FZIVSTA, MGIEGEL, MINDVAL, MJUSBAL, MPAUKRI, MVILRAS, MVYTUGI, MDOMZEM.*
3. *C aplankas*: *FAIRDAP. FASTTAU, FDOVSUK, FEDIANI, FERIDIL, FGIESTU, FINASAS, FINGMAL, FJOLPUN, FJOVITA, FOMEALB, FMARKIE, MRIMRAK, MROKKRI, MSIMGLI, MSTAGEL, MVALBUD, MANDVAL, MDONZUK, MEVAPOV*.

4. *D aplankas*: *FJURABE, FJURZAP, FJUSJOC, FJUSLIL, FKRISUK, FKRIVAI, FLINMAZ, FLOREBE, FMARJOC, FMARKOS, FMILKAR, FONAMAL, MKARDOV, MLUKJUO, MMARZIL, MMYKBAR, MNERDAU, MROBBIV, MSIMBUR, MTADCEK.*
5. *E aplankas: FAGNGRA, FAGNVIN, FAISIZI, FAISZYM, FAUSNEM, FDAILOI, FGINGED, FIEVJUR, FIEVSAB, FIEVVIS, FJUSKIN, FKAMMOS, FLAUZET, MDAIGUS, MEDGVOL, MKAZANU, MLINJUR, MMODSLE, MRIMAPA, MVYGVAI.*

Monofoninis metodas. Paleidžiamas metodinėje dalyje aprašytas monofoninio metodo *run.sh* skriptas. Gaunami automatinio kalbos atpažinimo testavimo rezultatai pateikiami lentelėse:

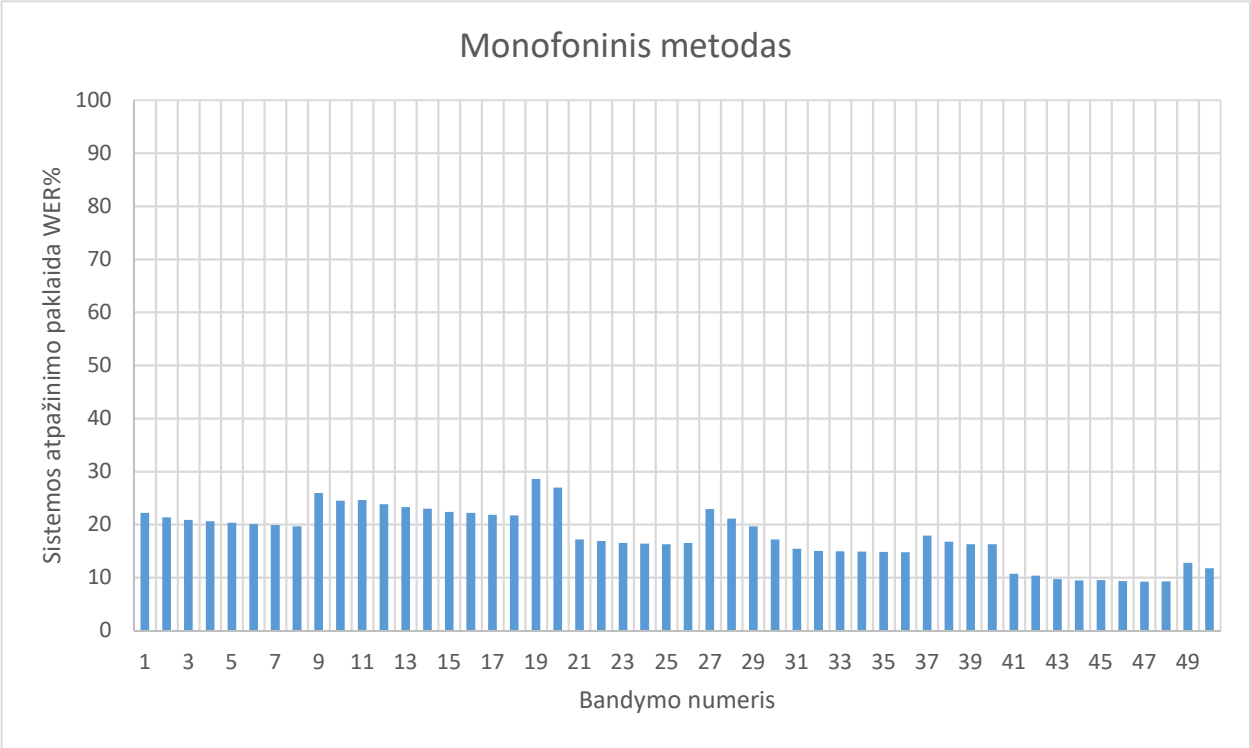
3 lentelė. Kryžminio bandymo rezultatai, pritaikius monofoninį akustinį modelį.

Aplankas/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	22,19	21,33	20,88	20,6	20,35	20,07	19,89	19,67	25,96	24,48
B-aplankas	24,58	23,83	23,28	22,98	22,40	22,20	21,85	21,7	28,61	26,94
C-aplankas	17,20	16,9	16,52	16,40	16,27	16,50	22,89	21,09	19,66	17,20
D-aplankas	15,43	15,02	14,98	14,88	14,82	14,8	17,92	16,75	16,27	16,27
E-aplankas	10,75	10,40	9,73	9,45	9,5	9,35	9,25	9,3	12,8	11,77

4 lentelė. Bandymų su monofoniniu akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	21,54	23,83	18,06	15,71	10,23
Geriausias	19,67	21,70	16,27	14,8	9,25
Blogiausias	25,96	28,61	22,89	17,92	12,80

Iš lentelių matyti, kad automatinio kalbos atpažintuvo sistemos tikslumas sumažėjo ~20 % panaudojant modifikuotą 100 diktorių garsyną. Sistemos su monofininiu akustiniu modeliu bandymų atpažinimo paklaida svyruoja nuo 9,25 % iki 28,61 %. Reikia atsižvelgti į tai, kad *E-aplanko* rezultatai gavosi geresni ir šiek tiek iškraipo bendrą tyrimų rezultatų vidurkį. Toliau pateikiamas grafinis rezultatų atvaizdavimas:



46 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant monofoninį akustinį modelį.

Trifoninis metodas. Toliau seka sistemos su trifoniniu akustiniu modeliu patikrinimas. Šiuo atveju *run.sh* skripte aprašytas akustinis modeliavimas susideda iš dviejų dalių: monofoninio akustinio modelio sudarymas, jo paruošimas trifoniniui akustiniui modeliui ir trifoninio akustinio modelio sudarymas. Matyti, kad sistemos sudėtingumas pritaikant kitą akustinį modelį auga. Paleidus *linux* terminale komandą *bash run.sh* gauti ir apdoroti rezultatai pateikiami lentelėse:

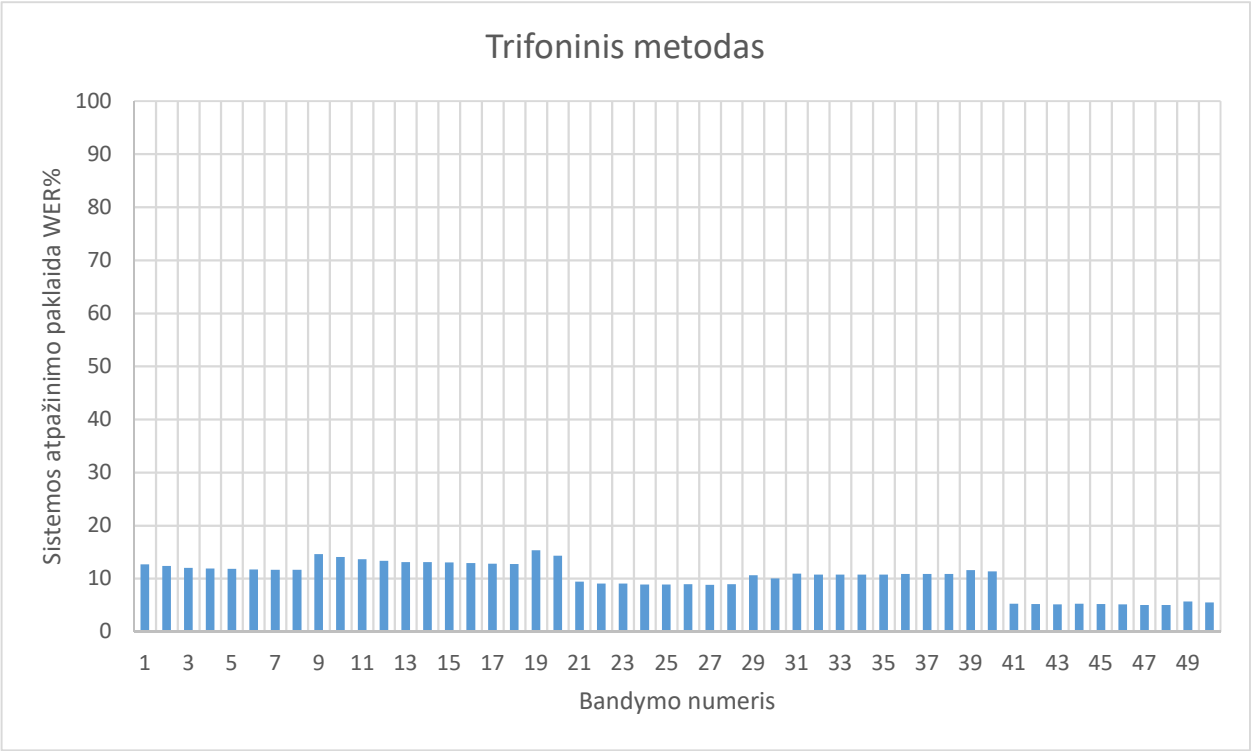
5 lentelė. Kryžminio bandymo rezultatai, pritaikius trifoninį akustinį modelį.

Folderis/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	12,67	12,36	12,04	11,91	11,86	11,74	11,66	11,68	14,58	14,05
B-aplankas	13,63	13,33	13,08	13,08	13,05	12,9	12,8	12,73	15,36	14,28
C-aplankas	9,43	9,05	9,05	8,85	8,88	8,95	8,80	8,93	10,63	10,03
D-aplankas	10,93	10,73	10,75	10,77	10,77	10,88	10,88	10,88	11,62	11,35
E-aplankas	5,28	5,22	5,15	5,25	5,22	5,15	5,03	5,00	5,65	5,50

6 lentelė. Bandymų su trifoniniu akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	12,45	13,42	9,26	10,95	5,24
Geriausias	11,66	12,73	8,80	10,73	5,00
Blogiausias	14,58	15,36	10,63	11,62	5,65

Kalbos atpažintuvo su *trifoniniu* akustiniu modeliu atveju matomas sistemos tikslumo pagerėjimas, kadangi trifoninio akustinio modelio apmokymo metu, sistema kaupia informaciją ne tik apie ištartą mažiausią rašto kalbos vienetą t.y. fonemą, bet ir tai kas yra tos fonemos aplinkoje. Šiuo atveju sistemos paklaidos diapazonas yra 5-15,36 %. Kaip minėta, *E-aplanko* rezultatai šiek tiek iškraipo bendrą tyrimų rezultatų vidurkį. Toliau pateikiamas grafinis *trifoninio* metodo bandymų atvaizdavimas:



47 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant trifoninį akustinį modelį.

LDA+MLLT metodas. *LDA+MLLT* metodas suklasifikuoja ir sumažina dimensijų skaičių apskaičiuotuose *Melų dažnių skalės kepstriniuose koeficientuose*. Iš to seka, kad sistema sugeba geriau išskirti būdingus vektorių požymius testuojamam ištarimui. Šiuo atveju reikalingi *trifoninio* akustinio modelio paruošti aplanko failai, tam kad apmokyti sistemą *LDA+MLLT* metodu. Gaunami rezultatai pateikiami lentelėse:

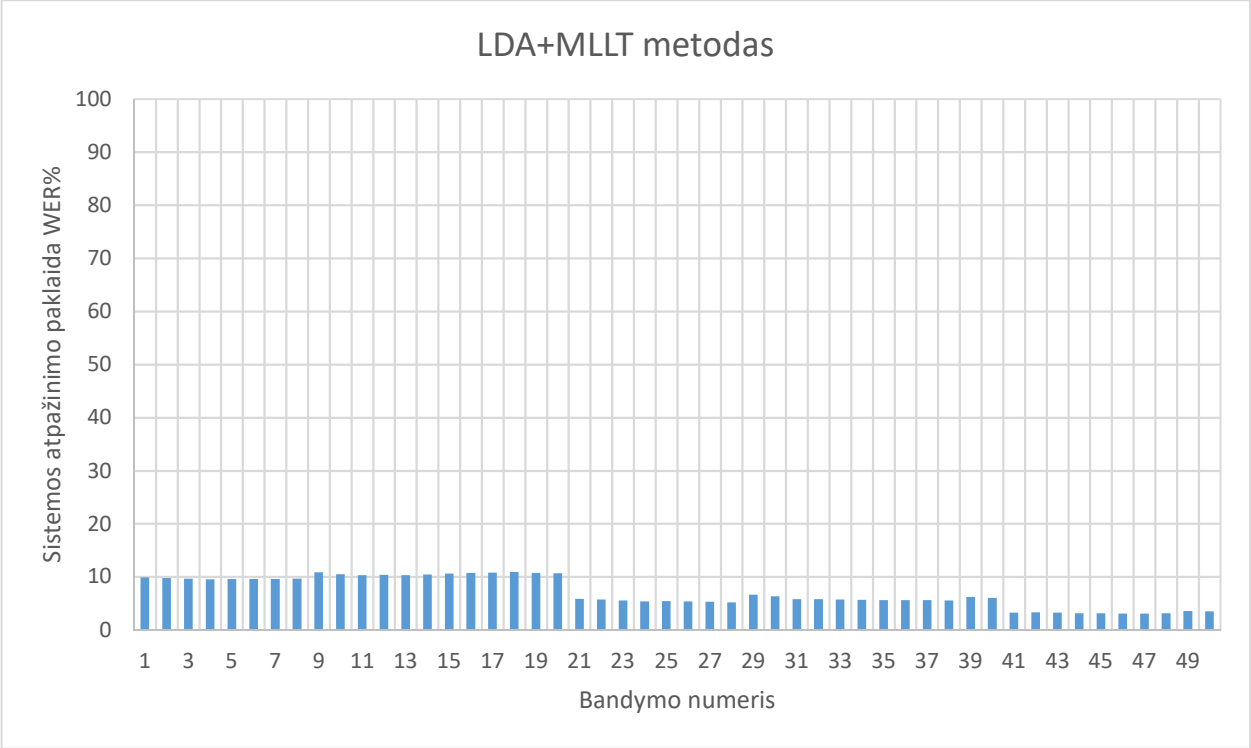
7 lentelė. Kryžminio bandymo rezultatai, pritaikius *LDA+MLLT* akustinį modelį.

Aplankas/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	9.92	9.75	9.64	9.52	9.59	9.57	9.57	9.67	10.88	10.48
B-aplankas	10.35	10.37	10.32	10.47	10.65	10.75	10.80	10.92	10.72	10.67
C-aplankas	5.89	5.74	5.54	5.37	5.42	5.39	5.32	5.19	6.62	6.34
D-aplankas	5.82	5.78	5.75	5.68	5.65	5.65	5.60	5.55	6.25	6.05
E-aplankas	3.28	3.30	3.28	3.17	3.15	3.10	3.10	3.15	3.58	3.53

8 lentelė. Bandymų su LDA+MLLT akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	9,89	10,59	5,72	5,78	3,27
Geriausias	9,52	10,32	5,19	5,55	3,10
Blogiausias	10,88	10,92	6,62	6,25	3,58

Pritaikius sudėtingesnį *LDA+MLLT* metodą, matomas sistemos tikslumo pagerėjimas – paklaida priklauso 3.1-10.92 % intervalui. Galima daryti prielaidą, kad papildomas *MFCC* koeficientų klasifikavimas pagerina bendrą kalbos atpažintuvo sistemos tikslumą. Geriausio bandymo rezultatas yra 3,1 % paklaida, blogiausio bandymo rezultatas – 10,92% paklaida. Toliau pateikiamas grafinis vaizdas su bandymų rezultatais:



48 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *LDA+MLLT* akustinį modelį.

LDA+MLLT+SAT metodas. Atlikus *MFCC* koeficientų klasifikavimą, taip pat, galima išskirti požymius, kuriuos lemia skirtingi diktorių balsai ir išskirti esminius parametrus pagal konkretų diktorių. Tam papildomai atliekamas *SAT* apmokymas. Pagrindiniame *run.sh* faile pridedamas kreipimasis į *train_sat.sh* skriptą. Gauti rezultatai pateikiami lentelėse:

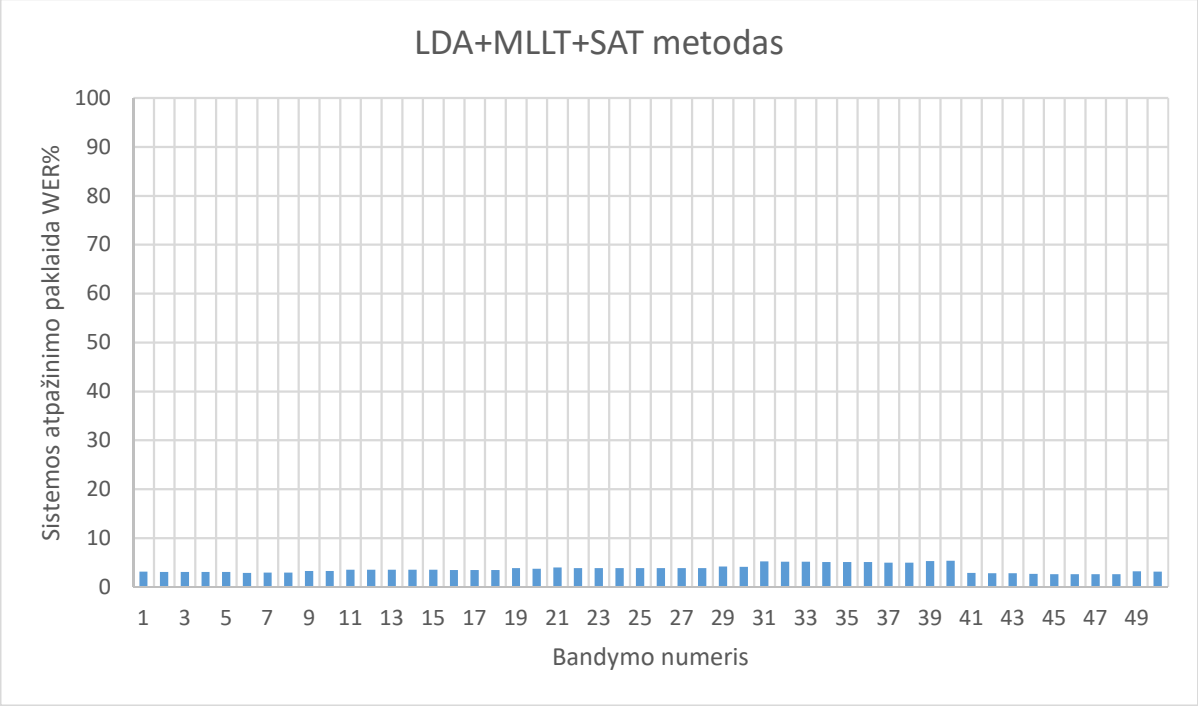
9 lentelė. . Kryžminio bandymo rezultatai, pritaikius *LDA+MLLT+SAT* akustinį modelį.

Aplankas/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	3,10	3,05	3,02	3,05	3,02	2,87	2,92	2,95	3,22	3,22
B-aplankas	3,51	3,51	3,53	3,51	3,48	3,46	3,46	3,43	3,81	3,68
C-aplankas	3,99	3,84	3,84	3,81	3,84	3,81	3,81	3,81	4,16	4,09
D-aplankas	5,22	5,12	5,12	5,10	5,07	5,05	4,97	4,95	5,30	5,32
E-aplankas	2,85	2,80	2,80	2,67	2,62	2,62	2,60	2,60	3,17	3,10

10 lentelė. Bandymų su *LDA+MLLT+SAT* akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	3,04	3,53	3,90	5,12	2,78
Geriausias	2,87	3,43	3,81	4,95	2,6
Blogiausias	3,22	3,81	4,16	5,32	3,17

Iš rezultatų matyti, kad *SAT* metodas, net ir kuriant atpažinimo sistemą nepriklausančią nuo konkretaus diktoriaus, pagerina atpažinimo tikslumą, bei sumažina neatpažintų komandų kiekį. Gauti rezultatai rodo, kad paklaidos vertė kinta nuo 2,6 % iki 5.32 %. Toliau pateikiamas gautų rezultatų grafinis vaizdas:



49 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *LDA+MLLT+SAT* akustinį modelį.

SGMM metodas. *SGMM* metodo principas yra turimo *HMM* modelio dimensijos tarpinės dimensijos sukūrimas. Šiuo atveju *SGMM* metodas atrenka tik labiausiai būdingus svertinus koeficientus ir rezultate gaunamas suspaustas turimo *HMM* modelio variantas. Tokiu atveju sistemos atpažinimo procesas tampa spartesnis, ir straipsniuose [14] [15] rašoma, kad sistemos tikslumas taip pat pagerėja. Gauti rezultatai su *SGMM* akustiniu modeliu pateikiami lentelėse:

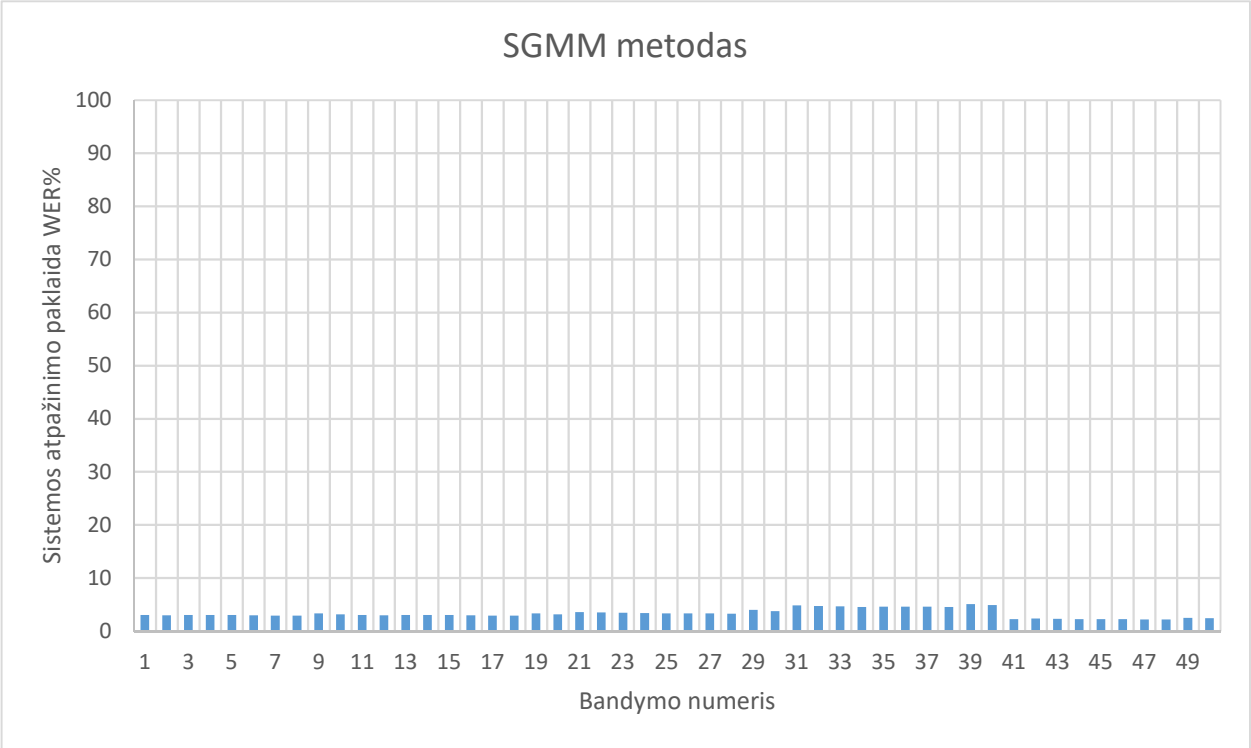
11 lentelė. Kryžminio bandymo rezultatai, pritaikius *SGMM* akustinį modelį.

Aplankas/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	3,03	2,98	3,01	3,01	3,01	2,96	2,93	2,93	3,31	3,16
B-aplankas	3,03	2,98	3,01	3,01	3,01	2,96	2,93	2,93	3,31	3,16
C-aplankas	3,59	3,49	3,46	3,39	3,34	3,34	3,31	3,28	4,01	3,74
D-aplankas	4,85	4,72	4,68	4,55	4,57	4,57	4,57	4,53	5,07	4,90
E-aplankas	2,23	2,33	2,28	2,25	2,22	2,22	2,17	2,2	2,45	2,4

12 lentelė. Bandymų su *SGMM* akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	3,03	3,03	3,49	4,70	2,27
Geriausias	2,93	2,93	3,28	4,53	2,17
Blogiausias	3,31	3,31	4,01	5,07	2,45

Matomas neryškus sistemos tikslumo pagerėjimas. Kalbos atpažintuvo paklaida yra 2,17-5,07 % diapazone. Galima daryti prielaidą, kad *SGMM* metodas nežymiai pagerina sistemos atpažinimo tikslumą, tačiau atsižvelgus į pateiktos komandos atitikmens suradimą, sistema sprendimą priima greičiau. Grafinis rezultatų vaizdas pateikiamas 50 paveiksle:



50 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *SGMM* akustinį modelį.

DNN metodas. Galiausiai, atliekamas kalbos atpažinimo sistemos apmokymas panaudojant neuroninį tinklą. Čia atlikus keletą išankstinių bandymų paaiškėjo, kad didžiausias tikslumas pasiekiamas naudojant 4 paslėptus sluoksnius, todėl kryžminiam patikrinimui, buvo parinkti 4 neuroninio tinklo sluoksniai. Remiantis teorija, neuroniniam tinklui apmokyti kaip pirminis akustinis modelis buvo pateikiamas *LDA+MLLT+SAT* modelis. Gauti rezultatai pateikiami lentelėse:

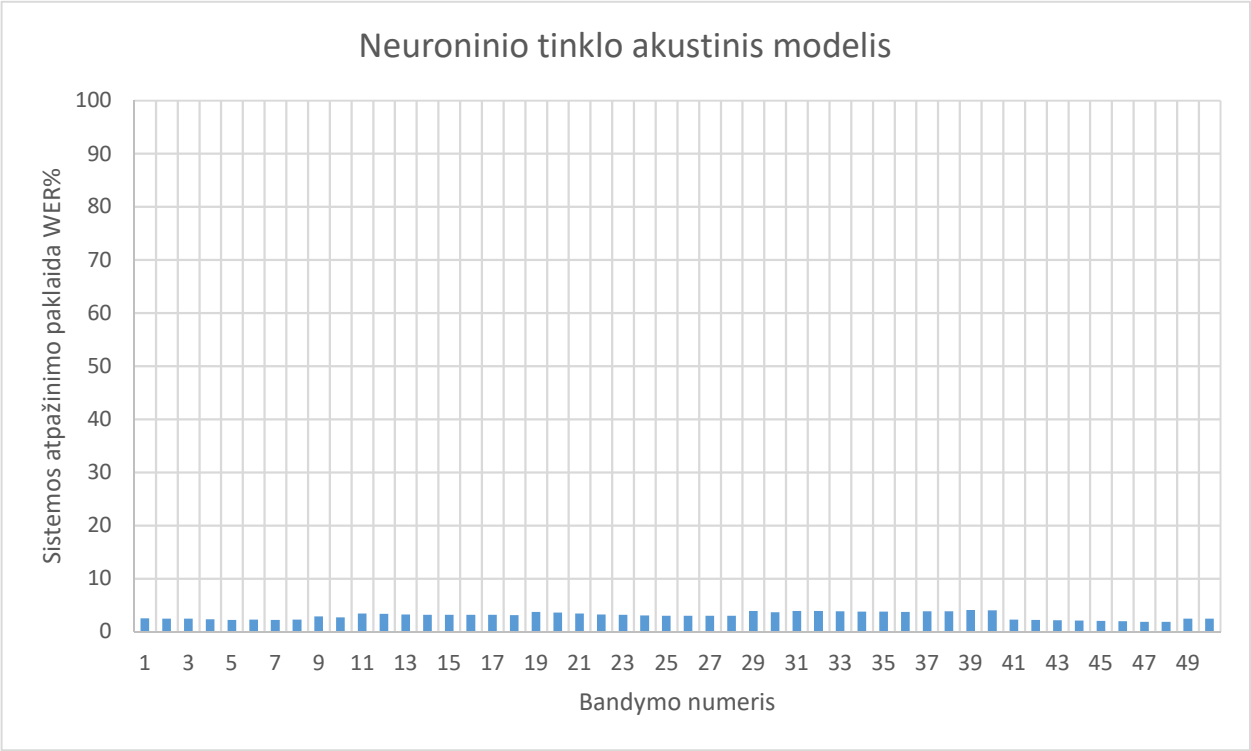
13 lentelė. Kryžminio bandymo rezultatai, pritaikius *DNN* akustinį modelį.

Folderis/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A-aplankas	2,47	2,44	2,42	2,32	2,22	2,24	2,22	2,24	2,85	2,67
B-aplankas	3,38	3,36	3,23	3,18	3,16	3,16	3,13	3,08	3,71	3,56
C-aplankas	3,41	3,21	3,13	3,01	2,98	2,96	2,96	2,96	3,91	3,66
D-aplankas	3,9	3,88	3,83	3,78	3,75	3,72	3,83	3,85	4,07	4,03
E-aplankas	2,28	2,20	2,12	2,05	2,03	1,95	1,85	1,83	2,45	2,45

14 lentelė. Bandymų su *DNN* akustiniu modeliu rezultatų suvestinė.

	A-aplankas	**B-aplankas**	**C-aplankas**	**D-aplankas**	**E-aplankas**
Vidurkis	2,40	3,29	3,21	3,86	2,12
Geriausias	2,22	3,08	2,96	3,72	1,83
Blogiausas	2,85	3,71	3,91	4,07	2,45

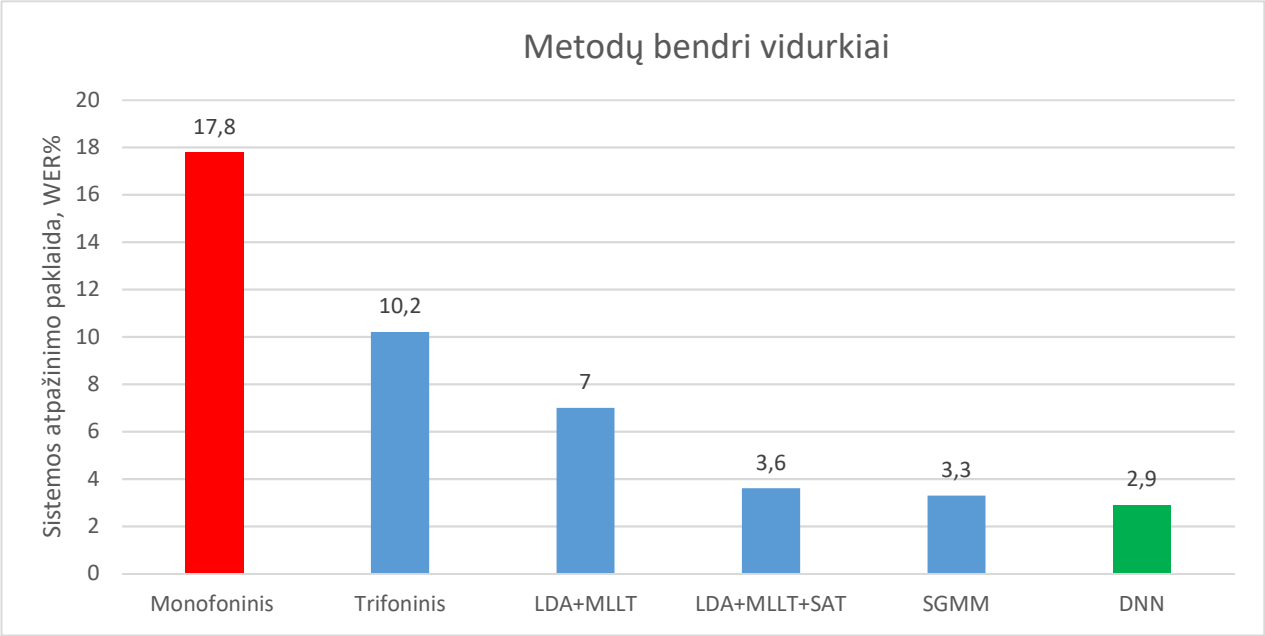
Sistemos su neuroniniu tinklu paklaida svyruoja tarp 1,83 % ir 4.07 %. Šiuo metodu buvo pasiektas tiksliausias sistemos atpažinimas. Toliau pateikiamas grafinis rezultatų atvaizdavimas paveiksle:



51 pav. Kryžminio patikrinimo bandymu rezultatai, naudojant *DNN* akustinį modelį.

3.3. Bendras 6 metodų palyginimas.

Aiškesniam gautų rezultatų atvaizdavimui, ordinačių ašies diapazonas sumažinimas (WER = 0-30 %). Visų atliktų metodų bandymų rezultatai pateikiami viename grafike:



52 pav. 6 metodų bandymų rezultatai viename grafike.

Iš grafiko matyti, kad kalbos atpažinimo sistemai, parinkus neuroninio tinklo akustinį modelį gaunama mažiausia sistemos atpažinimo paklaida (1,83-4,07 %, bandymų vidurkis 2,98 %). Šiek tiek prastesni rezultatai buvo gauti panaudojant *SGMM* (2,17-5,07 %, bandymų vidurkis 3,3 %) ir *LDA+MLLT+SAT* (2,6-5,32 %, bandymų vidurkis 3,6 %) metodus. Analizuojant šių trijų bandymų rezultatus pastebėta, kad gaunamas mažesnis paklaidos svyravimas tarp atskirų bandymų su skirtingais diktoriais. Galima teigti, kad *SAT* metodas, išskirdamas požymius, priklausančius nuo konkretaus kalbėtojo, pagerina sistemos stabilumą, taip pat sistema sugeba atpažinti įvairesnį žmogaus balso pobūdį tokiu pačiu tikslumu.

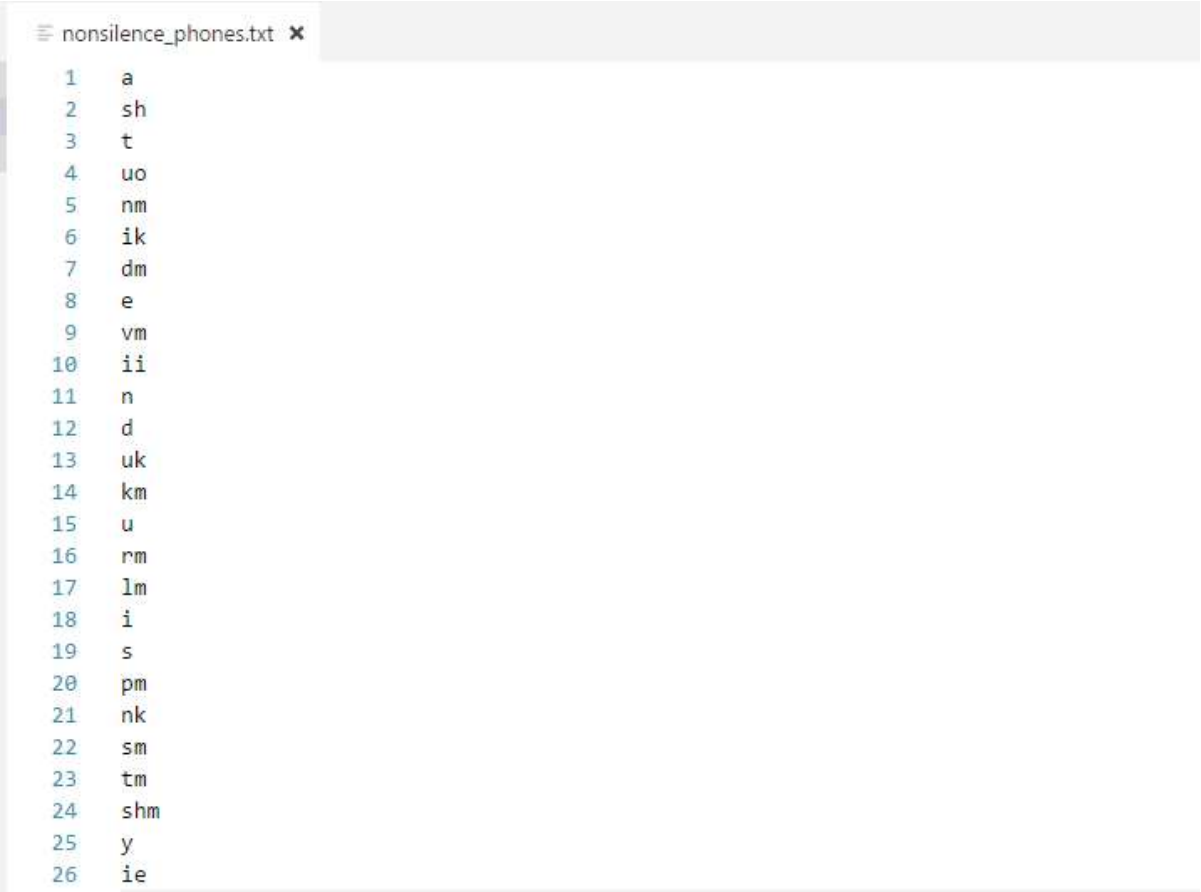
Taip pat stebimas ženklus sistemos atpažinimo paklaidos sumažėjimas pakeičiant *monofoninį* (9,25-28,61 %, bendras vidurkis 17,8 %) metodą *trifoniniu* (5-15,36 %, bendras vidurkis 10,26 %). Tinklo apmokymas, atsižvelgiant į šalia fonemos esančias kitas fonemas turi daug įtakos automatinio kalbos atpažinimo tikslumui.

3.4. Tyrimai su SAMPA-LT fonemų rinkiniu

Šiame poskyryje atliekami bandymai su pasirinktu vienu iš prieš tai aprašytu aplanku, panaudojant *SAMPA-LT* fonemų rinkinį. Bandymams parenkamas *B-aplankas*.

SAMPA (angl. *Speech Assesment Method Phonetic Alphabet*) yra kalbos modelio sudarymo metodas, skirtas tarptautinio alfabeto rinkinio simbolius paversti į *ASCII* kodavimą. *SAMPA-LT* yra lietuvių alfabeto rinkinys sukurtas pagal *ASCII* kodavimą. [22]

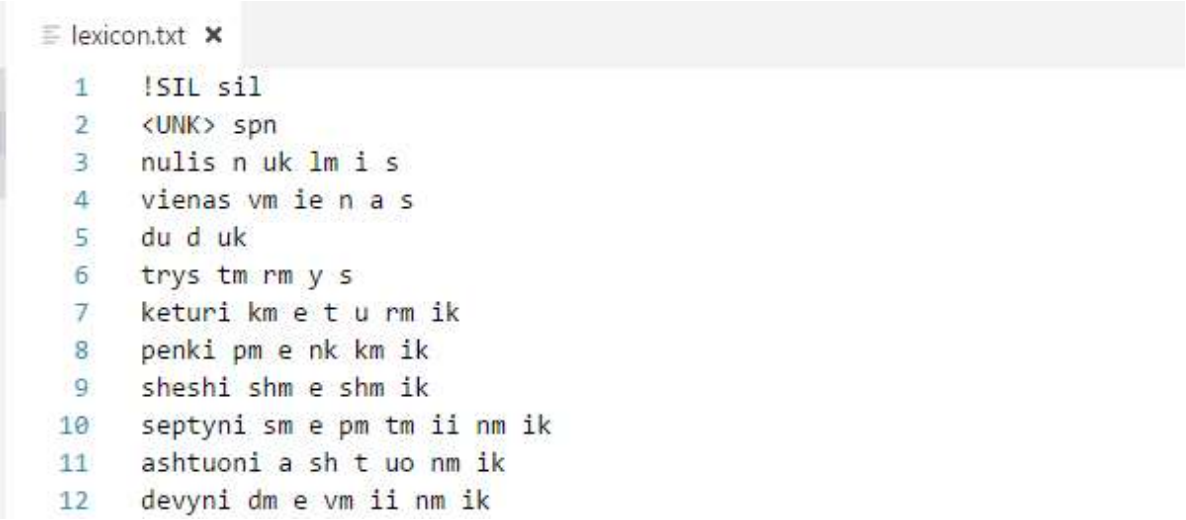
Norint atlikti bandymus su skirtingu fonemų rinkiniu reikia pakeisti bendrą fonemų sąrašo failą *nonsilence_phones.txt*. Toliau pateikiamas *nonsilence_phones.txt* failo turinys:



53 pav. *SAMPA-LT* fonemų rinkinio sąrašas.

Šį fonemų rinkinį sudaro 26 fonemos. Prie kiekvienos fonemos pridedamas kietumo (k) arba minkštumo (m) ženklas.

Taip pat reikia per naujo aprašyti visų galimų komandų failą *lexicon.txt*, panaudojant *SAMPA-LT* fonemų rinkinį:



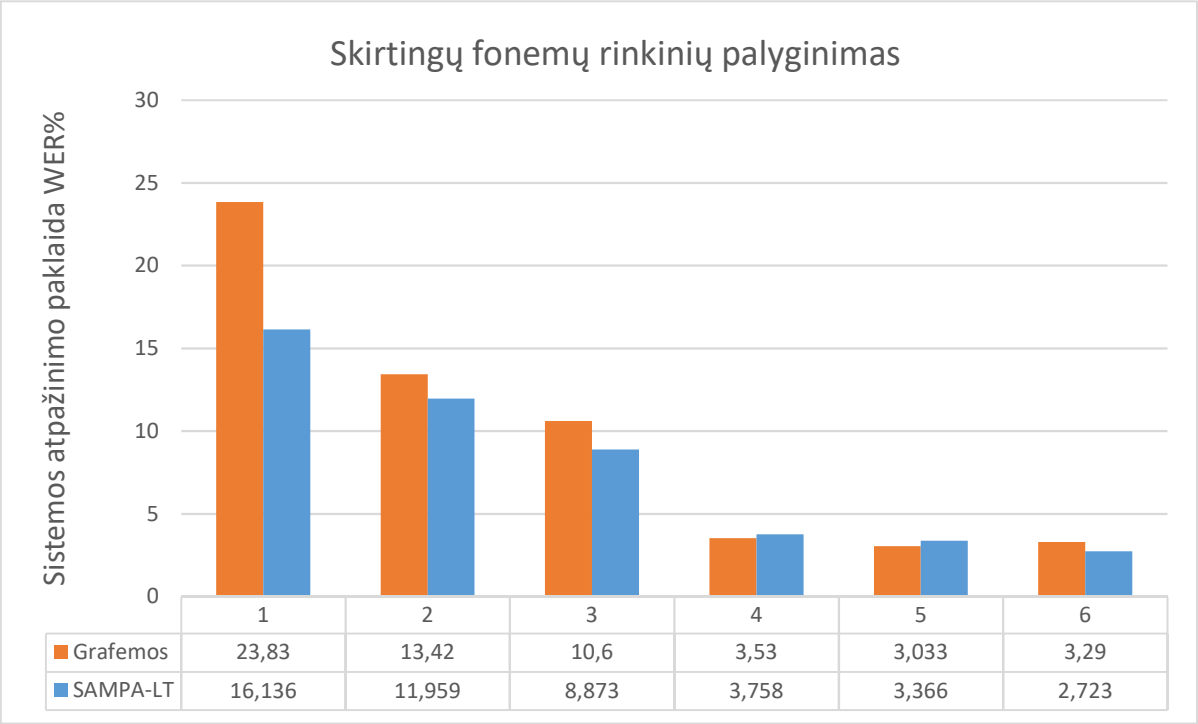
54 pav. *lexicon.txt* failas aprašytas panaudojant *SAMPA-LT* fonemų rinkinį.

Kaip ir minėta, tyrimai atliekami su *B-aplanku*, pritaikant 6 akustinio modeliavimo metodus. Pasinaudojant *bash run.sh* komanda, paleidžiamas *run.sh* skriptas. Toliau pateikiama bandymų rezultatų lentelė:

15 lentelė. *B-aplanko* bandymų rezultatai su *SAMPA-LT* fonemų rinkiniu, panaudojant 6 skirtingus modeliavimo metodus.

Metodas/bandymas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Monofoninis	15,74	15,53	15,51	15,43	15,46	17,46	16,91	16,31	16,31	16,70
Trifoninis	12,08	12,03	11,90	11,70	11,58	11,48	11,48	11,43	13,18	12,73
LDA+MLLT	9,15	9,02	8,72	8,57	8,52	8,52	8,47	8,44	9,87	9,45
LDA+MLLT+SAT	3,83	3,81	3,76	3,76	3,73	3,66	3,63	3,53	3,96	3,91
SGMM	3,38	3,36	3,36	3,33	3,31	3,31	3,33	3,31	3,51	3,46
DNN	2,76	2,71	2,66	2,61	2,63	2,66	2,63	2,68	2,98	2,91

Taip pat 55 pav paveiksle atvaizduojamas bendras kiekvieno metodo bandymų vidurkis, sulyginant su bandymų rezultatais, panaudojant grafemas:



55 pav. Skirtingų fonemų rinkinių bandymų rezultatai (1 blokas – *monofinis metodas*, 2 blokas – *trifoninis metodas*, 3 blokas – *LDA+MLLT*, 4 blokas – *LDA+MLLT+SAT*, 5 blokas – *SGMM*, 6 blokas – *DNN*).

Bandymų rezultatai rodo, kad sistemai su monofoniniu akustiniu modeliu pritaikytas *SAMPA-LT* fonemų rinkinys padidino tikslumą daug labiau, lyginant su sistemomis su kitais akustiniais modeliais. Pritaikius *SAT* metodą stebimas netgi atpažinimo pablogėjimas. Visais kitais atvejais, *SAMPA-LT* šiek tiek pagerina sistemos atpažinimo tikslumą.

Išvados

1. Sukurta darbinė *linux* aplinka ir atlikti bandymai su *Kaldi* programiniu paketu.
2. Panaudojus švaraus garso signalo 30 diktorių garsyną, gautas 99,75 % skaičių garsyno atpažinimo tikslumas. Remiantis straipsniu [1], atlikti bandymai, naudojant tokį patį *SKAIC30* diktorių garsyną kalbos atpažinimo sistemos tikslumas yra ~99 % - tiek *Kaldi* programiniu paketu, tiek *HTK*, gaunamas pakankamai geras rezultatas.
3. Gavus 99,75 % tikslumą, nuspręsta išplėsti turimą garsyną iki 100 diktorių, taip pat kiekvienam garso įrašui pritaikyti 5dB foninį triukšmą tam, kad būtų galima tirti sudėtingesnius kalbos atpažinimo metodus ir matytųsi, kaip kinta sistemos tikslumas.
4. Bandymams su 100 diktorių garsynu panaudoti akustinio modeliavimo metodai: *monofoninis*, *trifoninis*, *LDA+MLLT*, *LDA+MLLT+SAT*, *SGMM*, *DNN*. Didžiausias skaičių garsyno atpažinimo sistemos tikslumas buvo pasiektas naudojant *DNN* akustinio modeliavimo metodą – bendras rezultatų vidurkis ~97 %.
5. Tinklo apmokymas, atsižvelgiant į šalia fonemos esančias kitas fonemas turi daug įtakos automatinio kalbos atpažinimo tikslumui, naudojant *trifoninį* akustinio modeliavimo metodą.
6. Pritaikius *SAT* metodą, pastebimas sistemos stabilumo padidėjimas, atsižvelgiant į skirtingą diktorių balso toną, tembrą ir t.t.
7. Papildomai atlikti tyrimai su *SAMPA-LT* fonemų rinkinių. Tyrimų rezultatai rodo, kad vietoj grafemų panaudojus *SAMPA-LT* fonemų rinkinį, skaičių garsyno atpažinimo sistemos stabilumas pagerėja naudojant *monofoninį* metodą (nuo ~76 % iki ~84 %), *trifoninį* (nuo ~86 % iki 88 %) *LDA+MLLT* (nuo ~89 % iki 91 %) ir *DNN* (nuo ~96 % iki 97 %).

Literatūros sąrašas

1. Bartišiūtė, G., Paškauskaitė, G., Ratkevičius, K. (2014). *Investigation of disease codes recognition accuracy.* Kauno technologijų universitetas, Lietuva.
2. *A Complete Guide to Speech Recognition Technology* [interaktyvus]. 2018 [žiūrėta 2019-02-01]. Prieiga per: https://www.globalme.net/blog/the-present-future-of-speech-recognition#Amazon_Alexa
3. Young, S. J., & Young, S. (1993). *The HTK hidden Markov model toolkit: Design and philosophy*. University of Cambridge, Department of Engineering. [žiūrėta 2019-02-01]. Prieiga per: https://www.researchgate.net/profile/Steve_Young3/publication/263124034_The_HTK_Hidden_Markov_Model_Toolkit_Design_and_Philosophy/links/555dc15d08ae6f4dcc8c5b62/The-HTK-Hidden-Markov-Model-Toolkit-Design-and-Philosophy.pdf
4. Matarneh, R., Maksymova, S., Lyashenko, V., & Belova, N. (2017). *Speech recognition systems: A comparative review*. [žiūrėta 2019-02-15]. Prieiga per: http://openarchive.nure.ua/bitstream/document/6388/1/M1905047179.pdf
5. Open-Source Large Vocabulary CSR Engine Julius [interaktyvus]. [žiūrėta 2019-02-20]. Prieiga per: http://julius.osdn.jp/en_index.php
6. Saini, P., & Kaur, P. (2013*). Automatic speech recognition: A review*. International Journal of Engineering Trends and Technology,4(2), 1-5. [žiūrėta 2019-03-02]. Prieiga per: https://pdfs.semanticscholar.org/86d1/28fc7728e434d7978e8edb8a5c3c60321926.pdf
7. Dave, N. (2013). *Feature extraction methods LPC, PLP and MFCC in speech recognition. International journal for advance research in engineering and technology*, *1*(6), 1-4. [žiūrėta 2019-03-09]. Prieiga per: https://www.researchgate.net/profile/Namrata_Dave2/publication/261914482_Feature_extraction_methods_LPC_PLP_and_MFCC_in_speech_recognition/links/562dce4908ae04c2aeb4aa1b/Feature-extraction-methods-LPC-PLP-and-MFCC-in-speech-recognition.pdf
8. Driaunys, K. (2006). *Lietuvių šnekamosios kalbos segmentavimo ir fonetinio atpažinimo tyrimas naudojant ltdigits garsyno įrašus*: daktaro disertacija. Vilniaus universitetas.
9. PRANEVIČIUS, Henrikas ir kt. *Agentinių sistemų modeliai*. Kauno technologijois universitetas, 2008 [žiūrėta 2019-03-09]. Prieiga per: https://www.mif.vu.lt/katedros/cs/Asmen/Raudys_DaugiaagentesSistemos_3skyrius.pdf
10. Haeb-Umbach, R., & Ney, H. (1992, March). *Linear discriminant analysis for improved large vocabulary continuous speech recognition.* In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing* (Vol. 1, pp. 13-16). IEEE. [žiūrėta 2019-03-09]. Prieiga per: https://www.math.arizona.edu/~hzhang/math574m/Read/ldaFace.pdf
11. Balakrishnama, S., & Ganapathiraju, A. (1998). *Linear discriminant analysis-a brief tutorial*. Institute for Signal and information Processing, 18, 1-8. [žiūrėta 2019-03-09]. Prieiga per: http://www.music.mcgill.ca/~ich/classes/mumt611_07/classifiers/lda_theory.pdf

12. Olsen, P. A., & Gopinath, R. A. (2004). *Modeling inverse covariance matrices by basis expansion*. IEEE Transactions on Speech and Audio Processing, 12(1), 37-46. [žiūrėta 2019-03-09]. Prieiga per: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1261270
13. Anastasakos, T., McDonough, J., & Makhoul, J. (1997, April). *Speaker adaptive training: A maximum likelihood approach to speaker normalization.* In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing* (Vol. 2, pp. 1043-1046). IEEE. [žiūrėta 2019-03-10]. Prieiga per: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=596119
14. Povey, D., Burget, L., Agarwal, M., Akyazi, P., Feng, K., Ghoshal, A., ... & Rose, R. C. (2010). *Subspace Gaussian mixture models for speech recognition*. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 4330-4333). IEEE. [žiūrėta 2019-03-10]. Prieiga per: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5495662
15. Povey, D., Burget, L., Agarwal, M., Akyazi, P., Kai, F., Ghoshal, A., ... & Rose, R. C. (2011). *The subspace Gaussian mixture model—A structured model for speech recognition*. Computer Speech & Language, 25(2), 404-439. [žiūrėta 2019-03-10]. Prieiga per: https://www.sciencedirect.com/science/article/pii/S088523081000063X
16. Richardson, F., Reynolds, D., & Dehak, N. (2015). *A unified deep neural network for speaker and language recognition. arXiv preprint arXiv:1504.00923*. [žiūrėta 2019-03-18]. Prieiga per: https://arxiv.org/pdf/1504.00923.pdf
17. Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., ... & Sainath, T. (2012). *Deep neural networks for acoustic modeling in speech recognition.* IEEE Signal processing magazine, 29. [žiūrėta 2019-03-18]. Prieiga per: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/HintonDengYuEtAl-SPM2012.pdf
18. Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., ... & Silovsky, J. (2011). *The Kaldi speech recognition toolkit* (No. CONF). IEEE Signal Processing Society. [žiūrėta 2019-03-18] Prieiga per: https://infoscience.epfl.ch/record/192584/files/Povey_ASRU2011_2011.pdf
19. *Ubuntu oficialus tinklalapis* [interaktyvus]. [žiūrėta 2019-02-20]. Prieiga per: https://www.ubuntu.com/
20. *Oracle oficialus tinklalapis* [interaktyvus] [žiūrėta 2019-02-20]. Prieiga per: https://www.oracle.com/index.html
21. Apmokomasis tinklaraštis. *Kaldi for dummies.* [interaktyvus] [žiūrėta 2019-02-20]. Prieiga per: https://kaldi-asr.org/doc/kaldi_for_dummies.html
22. Raškinis, A, Raškinis, G., Kazlauskienė, A. (2003). *SAMPA (Speech Assesment Methods Phonetic Alphabet) for encoding transcriptions of Lithuanian speech corpora.* Vytautas Magnus University. [žiūrėta 2019-04-28]. Prieiga per: https://www.vdu.lt/cris/handle/20.500.12259/55530

Priedai

1 priedas. Java programos kodas aprašomiesiems garsyno failams generuoti

```
/.../
package kaldi;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;


public static void main(String[] args) {
    makeTxtFileWAV(new File("/home/anon/kaldi/egs/sk_24_6/sk_audio/test")
            ,new File("/home/anon/kaldifiles/wav.scp"), txtType: "WAV");
    makeTxtFileWAV(new File("/home/anon/kaldi/egs/sk_24_6/sk_audio/test")
            ,new File("/home/anon/kaldifiles/text.txt"), txtType: "TEXT");
    makeTxtFileWAV(new File("/home/anon/kaldi/egs/sk_24_6/sk_audio/test")
            ,new File("/home/anon/kaldifiles/spk2gender"), txtType: "SPK2GENDER");
    makeTxtFileWAV(new File("/home/anon/kaldi/egs/sk_24_6/sk_audio/test")
            ,new File("/home/anon/kaldifiles/utt2spk"), txtType: "UTT2SPK");
    makeTxtFileWAV(new File("/home/anon/kaldi/egs/sk_24_6/sk_audio/test")
            ,new File("/home/anon/kaldifiles/text"), txtType: "TEXT");
//      generateCorpus();
}
```

```java
/**
 * recursion(File, List<String>, String) method is used for generate txt files
 * @param directory - directory or file, which contains wav files
 * @throws Exception when fails to find, open or write to files
 */
public static void recursionWAV (File directory, List<String> list, String txtType) throws Exception{
    for(String s : directory.list()){
        File failas = new File(directory.getAbsolutePath() + "/" + s);
        if(failas.isDirectory() && !failas.getName().equals("NE")   //equals because of dictor contains "NE"
                && !failas.getAbsolutePath().contains("TAIP")){
            recursionWAV(failas, list, txtType);
        }
        if("audio/x-wav".equals(identifyFileTypeUsingFilesProbeContentType(failas))
                && !failas.getAbsolutePath().contains("MANO")
                && !failas.getAbsolutePath().contains("mano")
                && !failas.getAbsolutePath().contains("m m")){
            switch (txtType){
                case "WAV":
                    list.add(s.substring(0, s.length() - 4) + " " + failas.getAbsolutePath() + "\n");
                    break;
                case "SPK2GENDER":
                    String nameSurname;
                    nameSurname = failas.getName().substring(0, s.length() - 7);
                    System.out.println(list.contains(nameSurname));
                    if(!list.contains(nameSurname + " " + nameSurname.substring(0, 1).toLowerCase() +"\n")){
                        list.add(nameSurname + " " + nameSurname.substring(0, 1).toLowerCase() +"\n");
                    }
                    break;
                case "TEXT":
                    String nameSurname2;
                    String[] numberStrings = new String[10];
                    numberStrings[0] = "nulis";
                    numberStrings[1] = "vienas";
                    numberStrings[2] = "du";
                    numberStrings[3] = "trys";
                    numberStrings[4] = "keturi";
                    numberStrings[5] = "penki";
                    numberStrings[6] = "sheshi";
                    numberStrings[7] = "septyni";
                    numberStrings[8] = "ashtuoni";
                    numberStrings[9] = "devyni";
                    nameSurname2 = failas.getName();
                    Integer number = new Integer(nameSurname2.substring(7, 8)); //error is possible, surround try catch
                    list.add(nameSurname2.substring(0, nameSurname2.length() - 4) + " " + numberStrings[number] + "\n");
                    break;
                case "UTT2SPK":
                    String nameSurname3;
                    nameSurname3 = failas.getName();
                    list.add(nameSurname3.substring(0, nameSurname3.length() - 4)
                            + " " + nameSurname3.substring(0, nameSurname3.length() - 7) +"\n");
                    break;
            }
        }
    }
}
```

```java
/**
 * writeListToFile(List<String> list, File toFile) method is used for
 * creating or rewriting a txt file (toFile) of list data
 * @param list - array implements List interface
 * @param toFile - File object with location path and name
 */
public static void writeListToFile(List<String> list, File toFile){
    FileWriter fw;
    Comparatorius comp = new Comparatorius();
    Collections.sort(list, comp);
    try {
        fw = new FileWriter(toFile);
        BufferedWriter bw = new BufferedWriter(fw);
        for(String writeLine : list){
            bw.write(writeLine);
        }
        bw.close();
    } catch (IOException ex) {
        System.out.println("Can't create file and write to file");
        ex.printStackTrace();
    }
}
```

```java
public static String identifyFileTypeUsingFilesProbeContentType(File file){
    String fileType = "Undetermined";
    try{
        fileType = Files.probeContentType(file.toPath());
    } catch (IOException ioException){
        System.out.println(
            "ERROR: Unable to determine file type for " + file
                + " due to exception " + ioException);
    }
    return fileType;
}
```

```
public static void generateCorpus(){
    FileWriter fw;
    try {
        fw = new FileWriter("/home/anon/kaldifiles/corpus.txt");
        BufferedWriter bw = new BufferedWriter(fw);
        for(int i = 0; i < 2000; i++){
            bw.write("nulis\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("vienas\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("du\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("trys\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("keturi\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("penki\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("sheshi\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("septyni\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("ashtuoni\n");
        }
        for(int i = 0; i < 2100; i++){
            bw.write("devyni\n");
        }
        bw.close();
    } catch (IOException ex) {
        System.out.println("Can't create file and write to file");
        ex.printStackTrace();
    }
}
```

Verta paminėti, kad buvo pasinaudota *Java* bibliotekos sąsaja (interface) *java.util.Comparator* galutinių sąrašų rikiavimui pagal abėcelę. Jei aprašomųjų garsyno failų turiniai (*spk2gender* , *utt2spk*, *text*, *wav.scp*, *corpus.txt*) yra nesurikiuoti, *Kaldi* sistema bando pati išrikiuoti sąrašus, bet atsiranda tikimybė, kad gali pasimesti sąrašo duomenys.

Norint aprašyti komparatorių, reikia *implementuoti java.util.Comparator* sąsają ir užkloti metodą *public int compare(T o1, T o2)*, pagal norimą lyginimo kriterijų. Toliau pateikiama aprašyta *Comparatorius* klasė:

```java
/.../
package kaldi;

import java.util.Comparator;

public class Comparatorius implements Comparator<String> {

    @Override
    public int compare(String s1, String s2) { return s1.compareTo(s2); }

}
```