TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Silver Vapper

Murdkeele kirjakeelestamise metoodikad

Bakalaureusetöö (9 EAP)

Juhendaja: Heiki-Jaan Kaalep

Tartu 2017

Murdkeele kirjakeelestamis metoodikad

Lühikokkuvõte:

Bakalaureusetöös keskenduti erinevatele võimalustele, kuidas saaks murdekeelseid tekste tõlkida eesti kirjakeele vormi. Selle eesmärk on muuta murdetekstid kergemini töödeldavateks. Katsetatud lahendusi oli kolm: murdekorpuse abil lemmade tõlkimine, regulaaravaldiste abil sõnade teisendamine ning statistiline masintõlge. Töös antakse ülevaade Eesti murrete taustast, kirjeldatakse loodud tõlkelahenduste tööpõhimõtet, analüüsitakse saadud tõlketulemusi ning tuuakse välja aspekte, mida lahenduste puhul annaks parandada või edasi arendada.

Võtmesõnad:

Murded, kirjakeel, tõlkimine, regulaaravaldised, keeletehnoloogia, masintõlge

CERCS: P170

Dialect Translation methods

Abstract:

This Bachelor's thesis focuses on different solutions for translating texts in dialects to Estonian written language so that it would make materials in dialects more easily processable. There are three different solutions implemented in this thesis: translation of lemmas using a dialect corpus, transformation on words with regular expressions and statistical machine translation method. Thesis gives an overview of Estonian dialects, describes how created translation methods work, analyses gathered translation results and suggests aspects that could be improved or fixed.

Keywords:

Dialects, written language, translation, regular expressions, natural language processing, machine translation

CERCS: P170

Sisukord

Sissejuhatus

Antud bakalaureusetöö eesmärk on luua tarkvaraline lahendus, mis võimaldab murdkeelelisi tekste kirjakeelseteks teisendada. Enamikel eesti keele murretest puudub kindel kirjakeelne vorm. Seetõttu on ka enamik murrete üleskirjutisi üsna unikaalsed, sest need on valminud kirjapanijale omase stiili järgi. See olukord raskendab aga kirjapandud murdetekstide töötlemist. Süntaktiliselt erinevalt üles märgitud, kuid semantiliselt identset tähendust omavat sõna või fraasi on sel juhul väga keeruline leida.

Töö idee pärineb Eesti Kirjandusmuuseumilt. Neil on soov muuta oma murdekeeleliste muinasjuttude kogu kergemini töödeldavaks. Et seda saavutada on üheks võimaluseks juttudes leiduvaid murdesõnu ühisele baasvormile teisendada. Antud töös proovitakse seda lahendada kirjakeelde tõlkimise näol, katsetades selleks kolme erinevat meetodit.

Fookuses on peamiselt Võru ja Setu murded, sest kõige enam on Eesti Kirjandusmuuseumi poolt välja pakutud testmaterjalides just setokeelseid muinasjutte. Tulemusena piisab, kui suurem osa, kuid mitte kõik, sisendtekstis leiduvad sõnad, on suudetud kirjakeelsesse vormi viia. Seejuures ei ole peamine, et väljundteksti puhul säiliks täielik loetavus, vaid et lihtsustuks tekstide töötlemine, näiteks märksõnade otsingu näol.

Töö raames loodavad kolm lahendusmeetodit on murdekorpuse meetod, teisendusmeeteod ning statistiline masintõlke meetod. Esimesena mainitud lahenduse puhul püütakse igale sisendtekstis leiduvale sõnele leida sõnastikust tõlget. Üheks selliseks vahendiks, mis seda teha võimaldab, on Tartu Ülikooli ning eesti ja üldkeeleteaduse instituudi poolt loodud murdekorpus[1]. Teise meetodina mainitud teisendusreeglite lahendus üritab sõnu kirjakeelde teisendada läbi tähtede asendamise, lisamise ja eemaldamise operatsioonide. Kuna kõikide eesti keele murrete jaoks vajalike reeglite loomine on ülimalt mahukas ülesanne, siis antud töö raames luuakse vaid teatud osa neist, et näidata lahenduse võimalikkust. Kolmanda meetodina on töö raames katsetatud statistilise masintõlke algelist teostust, kasutades mudeli treenimiseks eesti-võru paralleelkorpuses leiduvaid tekste.

Töö esimene pool annab pikema ülevaate ülesande vajalikkusest ning Eesti murrete olemusest ja ajaloost. Sealjuures kirjeldatakse, mida murdetekstid endast täpsemalt kujutavad ning miks nende

[1] http://www.murre.ut.ee/mkweb/

kirjakeelde tõlkimine keerukaks osutub. Ka tuuakse näited sarnastest probleemidest ning nende lahendustest teiste keelte ja murrete baasil. Töö teises peatükis kirjeldatakse lähemalt valminud programme ning antakse ülevaade nende algoritmide töökäigust. Viimases peatükis tutvustatakse erinevate meetodite abil tehtud tõlketestide tulemusi nii üksikute tekstide näitel, kui Kirjandusmuuseumi muinasjuttude alusel. Lisaks on välja toodud kõigi kolme meetodi omapärad ning edasiarendusvõimalused. Lisade sektsioonis on testitud muinasjuttude originaalversioonid ning valminud programmide lähtekood, koos käivitus- ja kasutusjuhendiga.

Terminid

1. Murre ehk dialekt on piirkondlik eripärane keelekuju ning see erineb kirjakeelest ja teistest murretest sõnade häälikkoostise, sõnade muutmise, sõnavara ja lausestuse poolest [3].
2. Kirjakeel on keele vorm, mis on esitatud kirjasüsteemi abil [17].
3. Morfoloogiline analüsaator tuvastab suvalise tekstisõne kohta, mis on selle algvorm ja mis käändes või pöördes sõna parajasti võib olla [23].
4. Regulaaravaldis on sõne, mis kirjeldab või langeb kokku mingi sõnede hulgaga vastavalt kindlatele süntaksireeglitele ning mitmed tekstiredaktorid ja tööriistad kasutavad neid otsingute tegemiseks ja teksti käitlemiseks vastavalt etteantud mustritele [5].

1. Taustast

Antud peatükis tutvustatakse lähemalt Eesti murrete olemust, kuidas ja miks neid on tarvis säilitada, kuidas on murdeid tõlgitud mujal ning millised probleemid sellega kaasnevad.

1.1 Murdetekstide kogumine ning töötlemine

Eesti territoriaalpiirdesse jääb väga mitmeid murdeid. Osad neist on tuntumad ja neist kuuleb tihtipeale igapäevaelus või meedia vahendusel, nagu näiteks Võru või Saare murre. Osad on vähem levinud, sest neid kõneleda oskavate ja tahtvate inimeste arv on aastate jooksul vähenenud või on need tavakeelele niivõrd sarnased, et murdeomapärad ei paista välja. Olenemata, mis murdega on tegu, on neis aegade jooksul väljakujunenud palju lugusid, laule jms, mis on kohalikule kandile unikaalsed ja omapärased. Selleks, et taoline looming ei kaoks, on tarvis seda mingisugusel viisil talletada.

Üheks võimaluseks on seda teha helisalvestiste teel. Kuna enamikel murretel puudub ühtne kirjakeel või ülesmärkimise stiil, siis on pärimuskultuuri kogujatel kõige lihtsam ja täpsem neid just helisalvestistena koguda, seejuures säilitades loo kõige autentsemat vormi. Samas on helifailide formaadi puuduseks nende raske töödeldavus. Kuigi helisalvestistes paikneva info töötlemine, sellel otsingu teostamine jms on võimalik, siis pole see nii mugav ja kiire protsess kui näiteks tekstifailide puhul [21]. Seega ongi alternatiivseks võimaluseks kuuldu ja nähtu transkribeerida ning see kas paberkandjal või digitaalsete tekstide näol talletada.

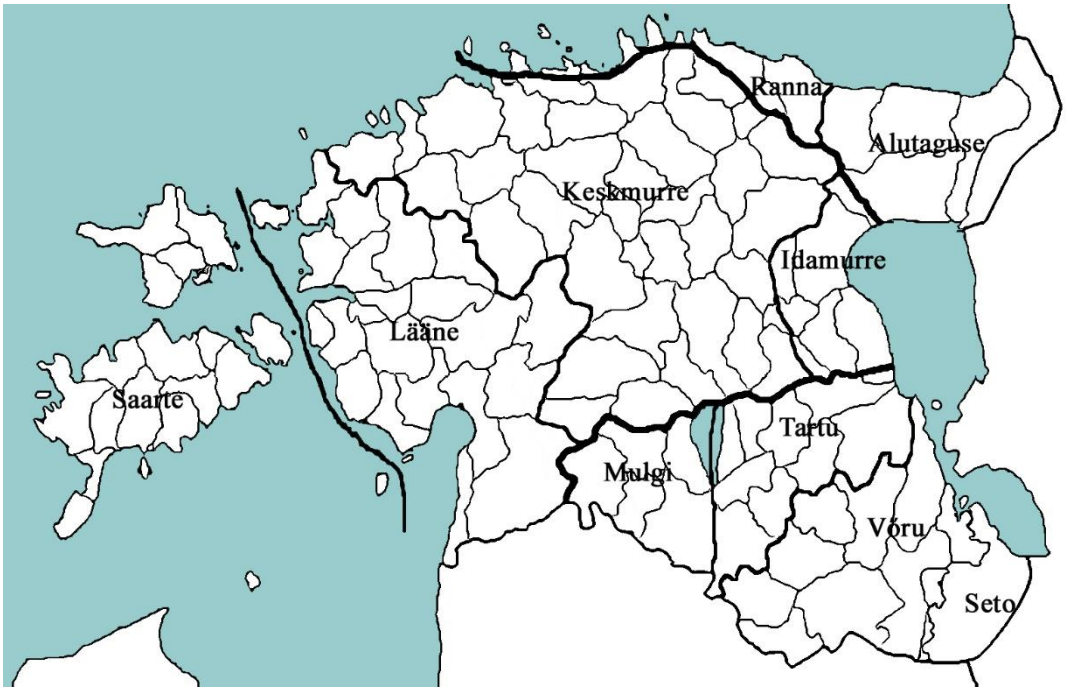
Ka tekstiliste materjalidega ei lahendu kõik probleemid. Ühtse kirjakeele puudumise tõttu on igal kirjapanijal oma nägemus, kuidas kuuldut üles märkida. Ka võib loo jutustaja häälda asju omapäraselt, mis erineb üldisest murdest. Taoline ebakõla toob kaasa situatsiooni, kus talletatud materjalid on lahus nii üksteisest, kui ka eesti keelest. Kui aga soovida viia läbi sisulist analüüsi kõikide eesti kultuuripärandis leiduvate materjalide kohta, on see taoliste materjalide puhul ülimalt keerukas, kui pea mitte võimatu ülesanne. Taolise olukorra leevendamiseks ehk teksti normaliseerimiseks [7] on mitmeid lahendusi:

- tõlkida kõik materjalid eesti kirjakeelde,
- püüda ühe murde siseseid ülesmärkimise erinevusi ühtlustada.

Antud töö raames proovitakse probleemile lahendust leida materjalide kirjakeelde tõlkimise teel. Taoline lähenemine võimaldab teoorias laiendada töödeldavate tekstide hulka ühe murde seast mitme murde vaheliseks.

1.2 Murded ja murdekeel

Eestis leiduvatel murretel on kõigil ühine algkeel, mis pärineb läänemeresoome keeltest, nagu on kirjutanud A. Kask oma raamatus „Eesti murded ja kirjakeel“ [10]. Nende tekkimise peamiseks põhjuseks on kunagine paikne ja kogukondlik eluviis, samuti ka traditsioon abielluda kihelkonna siseselt. 14.–15. sajandil valitses Eestis sunnimaisus, mis veelgi enam vähendas inimeste kaugemat läbikäimist ja suhtlemist. Seega juurdusidki tol perioodil kihelkondadele omapärased keelekujud, millest kasvasid välja murrakud. Murde levikualad on üldjoontes piiritletud ka kunagiste kihelkonna piiridega (vt joonis 1), millest enamasti tuleneb ka konkreetse murde nimi.

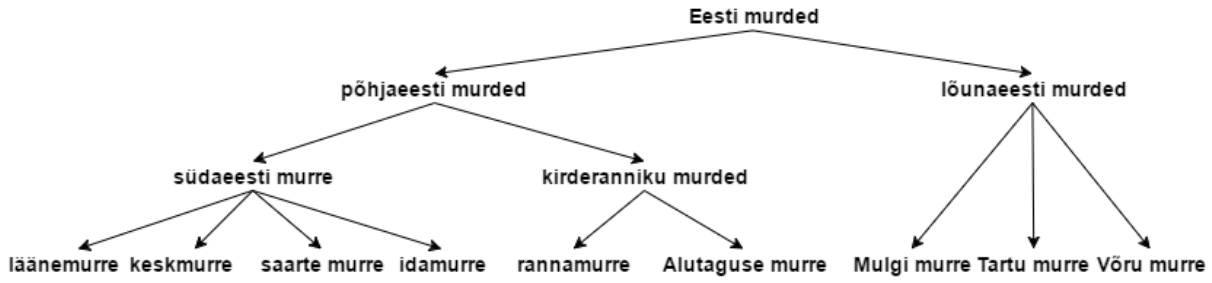


Joonis 1. Eesti murrete piirkondi kujutav kaart [14].

K. Pajusalu jt on lähtunud liigitusest [16], mille alusel jagunevad Eestis murded kahte suurde alarühma: põhjaeesti ja lõunaeesti murded.

Põhjaeesti murre liigitub veel omakorda südaeesti ja kirderanniku murreteks. Südaeesti murrete alla kuuluvad läänemurre, keskmurre, saarte murre ja idamurre. Kirderanniku murrete alla

kuuluvad rannamurre ja Alutaguse murre. Lõunaeesti murre jaguneb Mulgi murdeks, Tartu murdeks ja Võru murdeks. Murrete jagunemine on kujutatud ka joonisel 2.



Joonis 2. Eesti murrete üks võimalik jagunemine

Viimase sajakonna aasta jooksul on aga mitmed murded hääbuma hakanud, kuna inimeste liikumis- ja suhtlemisvõimalused on tänu tehnoloogia arengule jõudsasti lihtsamaks muutnud ning keskmurdest arenenud Eesti normeeritud kirjakeel on saanud enamike jaoks peamiseks kasutuskeeleks [10,17]. Saaremaa ning Lõuna-Eesti, eesotsas Võru- ja Setomaaga on jäänud veel kohtadeks, kus murdeline keelekasutus on igapäevaselt säilinud [12]. Nende murrakute püsivus tuleneb nende piirkondade paigutusest. Saartel elav rahvas on paratamatult teatavas geograafilises isolatsioonis, mis vähendab potentsiaalset igapäevast suhtlust Mandri-Eestis elava rahvaga. Võru murraku püsivus on samuti vähemalt osaliselt tingitud geograafilisest olustikust. See on Eesti maismaa osa, mis asub Põhja-Eestist kõige kaugemal ning seega omab vähem kokkupuudet keskmurdega.

Murrete omapärad

Eesti Keele Käsiraamatu järgi [3] on põhjaeesti ja lõunaeesti murderühmade erinevused peamiselt seotud keele ehitusega, näiteks käändsõna mitmusvormide konstrueerimine käib neis teisiti. Põhjaeesti murdeis kasutatakse te-mitmust kõigis käänetes alates sisseütlevast (*kala/de/st*), lõunaeesti murdeis aga vokaalmitmust (*kallu/st*). Rannikumurdele on omane sise- ja lõpukao puudumine mitmetes vormides (*nt kandama* pro *kandma, metsa* pro *mets*) ning ka vältevahetuse puudumine. Saare murre on tuntud oma *ö* tähe kasutamisele *õ* tähe asemel (*söber* pro *sõber*), läänemurre aga *b* esinemisele *v* asemel (*kibi* pro *kivi*). Idamurdes aga säilib *d* laadimuutuslike sõnade nõrgas astmes (*padas* pro *pajas).* Kuigi keskmurre on kirjakeelele kõige sarnasem, kuna see on kirjakeele aluseks, siis sellegipoolest on sellel erinevusi, nagu näiteks diftongide esinemine pikkade vokaalide asemel (*moa* pro *maa, miel* pro *meel).*

Lõunaeesti murderühma alla kuuluvatel murretel on samuti omapärased jooned. Mulgi murdes on *e* asendamas *a-d* juhul, kui see asub teisest silbist kaugemal (*kirjuteme* pro *kirjutame).* Tartu murdes esineb tugevaastmeline vokaalimitmus (*lehtist* pro *lehtedest* või *lehist).* Kõige rohkem erineb kirjakeelest Võru murre, millel on väga mitmeid iseäralike keelekonstruktsioone. Näiteks mõned nendest on *he*-tunnuseline sisseütlev kääne (*huunõhe* pro *hoonesse*), *h*- või *hn*-tunnuseline seesütlev kääne (*mõtsah, mõtsahn* pro *metsas*), järjekindel vokaalharmoonia (*kõnõlõm*a pro *kõnelema*), larüngaalklusiili esinemine (*naisõ?* pro *naised*, *kastõ?* pro *kaste*) [3].

1.3 Sarnased võõrkeelsed lahendused

Taoline probleem nagu normaliseerimata keele töötlemine on ka teistes keeltes peale eesti keele probleemiks. Näiteks Aasiast pärinevatel keeltel on sõltuvalt regioonist kümneid kui mitte sadu erinevaid dialekte, mida teatud inimgrupid igapäevaselt kasutavad [1,19]. Kaasajal on sotsiaalmeedia tõusuga suurenenud internetis tekstiline suhtlus ning üleüldine teksti kirjutamine. Üha tihedamini kasutatakse selleks ka oma keelekasutuses olevat dialekti, millel puudub selgelt struktureeritud reeglistik. Et aga taolises vormis veebi talletatud informatsiooni oleks võimalik keeletehnoloogiliste vahenditega töödelda, on vaja seda standardsesse kirjakeelde teisendada.

Näiteks on taolist probleemi üritatud lahendada tamili keele puhul Marimahtu K. ja S. L. Devi poolt tehtud uurimistöös [13]. Selle eesmärk oli välja selgitada ning hinnata tamili keele dialektide tõlkimisvõimalikust, kasutades lõplikke olekumasinaid (*finit state transducer*). Lõplike olekumasinate populaarsus on keeletehnoloogia vallas tõusvas trendis ning M. Hulden arvab selle taga olevat regulaaravaldiste kasutusvõimaluste suurenemist ja abstarktsioonitaseme laienemist [7]. Kuna tamili on aglutinatiivne keel ehk sõnavormid on moodustunud peamiselt morfeemide liitmise teel, siis on dialektidest tulenevad erinevused peamiselt tingitud sufiksite erikujudes. Lahenduse tarbeks koostati käsitsi automaadid, mis teisendavad murretest pärit sufiksid kirjakeelsete sufiksite vastu. Koostatud meetod andis positiivseid tulemusi ning sooritas teisendusi 85% täpsusega, küll aga toetas lahendus vaid viit väljavalitud dialekti.

Ka araabia keele põhjal on dialektide tõlkimisega tegeletud. H. Sawaf uurimistöö [18] seisnes hübriid masintõlke meetodi loomisel, mis suudaks veebi ja uudiste tekste tõlkida Egiptuse ning Levanti dialektidest Araabia kirjakeelde (ingl. MSA ehk *Modern Standard Arabic*). Hübriidmeetod sisaldas endas nii reeglipõhist lähenemist kui statistilist masintõlget, et

kompenseerida mõlema meetodi puuduseid. Töös tuuakse välja ka araabia dialektide normaliseerimislahendus, mis kasutab murdepõhiseid morfoloogilisi analüsaatoreid ja dekoodereid, et sisendtekstist standardiseeritud väljund saada. Varasemalt normaliseeritud dialektiteksti masintõlkimine MSA-sse andis 1-2% parema tulemuse.

Eelnevalt refereeritud Helsingi ülikooli professor M. Huldlen on koos kaasteadlastega uurinud ka baski keele dialektide ühtlustamist standardseks baski keeleks [6]. Nende tõlkemeetod tugines samuti lõplikel olekumasinatel, kuid selle asemel, et vastavaid teisendusi käsitsi kirja panna, üritasid nad neid luua automaatselt paralleelkorpuse põhjal. Kuna Baski dialektid on sõnade järjekorralt baski keelega enamasti sarnased, siis sai korpuses olevate lausete vahel teha üksühese joonduvuse (*alignment*) ehk iga sõnale lauses vastas tõlke lauses samal positsioonil olev sõna. Seejärel leiti sõnad, mis üksteisest erinesid ning nende põhjal genereeriti teisendus reeglid ning nendest omakorda lõplikud olekumasinad.

Antud lõputöös kasutatakse väljatoodud meetoditest mõneti erinevaid lähenemisi, mida täpsemalt kirjeldatakse peatükkis 2, kuid leidub ka sarnasusi. Tamili ja baski keele lõplike olekumasinate abil loodud teisendustele sarnaneb töö raames loodud teisendusmeetod, mis samuti muudab sõnas leiduvaid murdeerisusi neid asendades. Võrreldes baski dialektide jaoks loodud lahendusega ei ole teisendusmeetod aga kaugeltki nii automaatne. Samas on baski keele dialektid ja kirjakeelel omavahel ka väga sarnased. Vaid 23% korpuses olevatest sõnadest olid standardses baski keelest erinevad [6]. Seega Setu ja Võru murde jaoks oleks taoliste automaatide genereerimine keerukam, kuna nendes esineb nii sõnade järjekorra muutust kui tunduvalt rohkem erisusi sõnapaaride vahel. Araabia dialektidega tegelenud hübriidmasintõlke meetodi osaks on statistiline masintõlge, mida on ka antud töö raames proovitud Võru murde tõlkimisel rakendada.

1.4 Testitav tekstimaterjal

Käesoleva töö üheks konkreetseks ja mõõdetavaks eesmärgiks on tõlkida ära Eesti Kirjandusmuuseumile kuuluvad murdekeeles muinasjutud. Antud kogu on 11 000 eksemplari suurune ning sisaldab endas mitmete erinevate murrete ja murrakute tekste. Tekstid on talletatud .docx tüüpi failidesse. Metaandmete lisamiseks on iga teksti algusesse loodud tabel, milles on toodud välja ka teksti päritolu kant. Lisaks on seal märgitud teksti jutustaja ja üleskirjutaja nimed

ning salvestamise ning transkribeerimise kuupäevad. Kuna aga tekste on aastate jooksul väga mitmed erinevad inimesed kirja pannud, siis võivad ka ühes ja samas murdes olevad tekstid erinevalt üles märgistatud olla.

2. Tõlkimismeetodid

Selleks, et murdekeelseid tekste kirjakeelde teisendada, on antud töö raames proovitud kasutada kolme lähenemist. Esimene neist on loodud Tartu Ülikooli eesti ja üldkeeleteaduse instituudi ja Eesti Keele Instituudi koostöös valminud murdekorpuse[2] põhjal. Teine lähenemine on regulaaravaldiste abil loodud teisendused. Need on igale murdele omapärased reeglikogumikud, mis proovivad teisendada sõna murdevorme kirjakeelsesse vormi. Kolmanda lahendusena on katsetatud algelist statistilist masintõlget eesti ja võru keele tekstide paralleelkorpuste peal. Järgnevatest alampeatükkides tutvustatakse igat lahendust täpsemalt ning antakse konkreetsetest algoritmidest ülevaated.

2.1 Ühisjooned ja eeltöötlus

Murdetekstide tõlkimiseks on loodud erinevaid lahendusviise ja neil on ühiseid eeldusi, mida kõik neist suuremal või vähemal määral kasutavad. Näiteks on kõik tõlkimismeetodid kirjutatud programmeerimisekeeles Python, versioonis 3.5.2. Lisaks on kasutusele võetud mitmed teegid, et teostada tekstitöötlust, lugeda spetsiifilisi failitüüpe, nagu näiteks .docx või .cvs, teha veebipäringuid ning kasutada regulaaravaldisi. Üheks olulisimaks väliseks teegiks on aga Tartu Ülikooli teadlaste, eesotsas vanemteadur Sven Lauri poolt loodud Pythoni teek EstNLTK. Tegemist on keeletehnoloogilise teegiga, mis koondab endas mitmeid põhilisi ja olulisemaid funktsionaalsusi, milleks on: teksti tükeldamine sõnadeks ja lauseteks, morfoloogiline analüüs ja süntees, sõnade lemmatiseerimine, osalausestamine, ajaväljendite tuvastamine, nimeüksuste tuvastamine, verbiahelate tuvastamine, Eesti Wordneti liidestamine [4]. Antud töös kasutatakse EstNLTK teeki tekstide lemmatiseerimiseks, lausestamiseks ja sõnastamiseks.

Kõik töö raames tekstitõlkimiseks loodud meetodid on ülesehitatud sisendteksti tükeldamisele ja üksikute sõnade või lausete kaupa töötlemisele. Iga programmi alguses võetakse kasutaja poolt ette antud tekst ning muudetakse see EstNLTK vastavaid funktsionaalsusi kasutades väiksemateks osadeks, sõltuvalt vajadusest kas lauseteks või sõnadeks. Kui konkreetne meeteod on oma tõlkeprotsessi lõpetanud, pannakse saadud tulemustest kokku väljundtekst ning see kuvatakse kasutajale või kirjutatakse väljundfaili.

[2] http://www.murre.ut.ee/mkweb/

Murdekorpuse meetodi ja teisendusmeetodi ühisjooneks on ka see, et mõlemad loevad programmi käivitamisel sisse „Eesti keele seletava sõnaraamatu“ lemmadest koostatud faili ning genereerivad selle põhjal lemmade hulga, mille abil on teatud määral võimalik kontrollida sõnade kirjakeelsust. Seda rakendatakse mõlemas meetodis, et filtreerida välja sisendtekstist sõnad, mis ei vaja ümbertõlkimist. Antud meetodi puudujääk on nn vale-positiivsete juhtude esinemine, kus tunnistatakse kirjakeelseks sõnu, mis näivad kirjakeelsetena, kuid on siiski mõnes murdes hoopis teise tähendusega murdesõnad. Sellisteks juhtudeks on näiteks sõna *äi*, mis seletava sõnaraamatu järgi on mehe-või naiseisa, kuid murdes võib tähendada hoopis eitussõna *ei*.

Kõik selle töö raames valminud programmid on loodud käsurea rakendustena. Nende käivitamise täpsem juhend on antud töö lisas 1. Sisendtekste omistatakse programmile käsurealt. Programm võtab oma sisendiks kas .txt või .docx formaadis tekstifaili ning murde, milles tekst on loodud. Juhul kui tekstfaili ei lisata, siis on võimalik kasutajal käsurealt ka teksti sisestada, mida programm seejärel tõlkima hakkab. Eraldiseisev tugi on loodud Kirjandusmuuseumi poolt pakutud testmaterjalide sisselugemiseks, kuna antud failides on spetsiifiline struktuur ning tekstide vahele on paigutatud metaandmetega tabelid, kus on toodud tekstide päritolu kant jm info. Selle abil on võimalik anda hinnang, mis murdega võib tegu olla ning seeläbi murde määramist automatiseerida.

Tekstfailide töötlemiseks on loodud eraldi funktsioonid. Tavaliste .txt formaadis failide töötlemiseks on Pythonil endal vastav võimekus sisse ehitatud ning sisendväljund operatsioonidega saab teksti vähese vaevaga eraldatud. Microsoft Wordi .docx formaadi jaoks tuleb kasutada aga eraldi teeki, mis võimaldab antud formaadis faile parsida.

2.2 Murdekorpuse meetod

Antud meetod on töö jooksul loodud kolmest meetodist kõige laiaulatuslikum ehk katab kõige enam murdeid, kuna selle aluseks olev murdekorpus on suuteline eristama ühteteistkümmet eri murret ning veelgi enam murrakuid. Selle näol on tegemist andmebaasiga, mis hõlmab endas vanu murdetekste ja helisalvestisi. Enamik neist salvestistest pärinevad aastatest 1960-1970 [14].

Algoritmi kirjeldus

Antud lahendus filtreerib esmalt välja sõnad, mis kuuluvad tõlkimisele. Selleks, et hilisemalt oleks võimalik töödeldud tekstist uuesti ühtne tervik kokku panna, lisatakse originaalteksti märgendid.

Seejärel võetakse ette tõlkimisele kuuluvad sõnad ning nende alusel tehakse päring murdekorpuse veebiliidesele[3].

Iga sõna tagastab CSV-faili, kus on kirjas kõik tõlked, mis vastavale murdesõnale leiduvad. Vastav CSV-fail tuleb eraldi teegi abil läbi töödelda ning leida sealt soovitud murdele vastavad tõlked. Juhul kui sõnale leidub mitu sobivas murrakus olevat tõlget, kogutakse need kõik kokku ning leitakse neist enim esinenud variant. Miks ühele sõnale võib mitu ühes ja samas murdes olevat tõlget leiduda, on tingitud sellest, et murdekorpus tagastab sõnadele ka eri murrakutes tõlkevorme. Sobiva tõlke ümber lisatakse „$“ märgendid.

Kui päringule vastatud tulemustest ei leidu ühtki ostitud murdes olevat tõlget, püütakse sõna siiski tõlkida ülejäänud murrete alusel. Samuti leitakse üles maksimaalselt esinenud tõlge, mis annab suurima tõenäosusega sobilikuma variandi. Sel juhul märgistatakse tõlge küll eraldi tähistusega „@“, et väljundis oleks potentsiaalne valetõlge kergemini tuvastatav. Juhul kui sõnale ei leidu ühtegi sobivat tõlget, jäetakse sõna algsesse seisu ning märgistatakse tähistusega „#“, et see oleks hiljem väljundis eristatav.

Kui sõna on tõlkeprotsessi läbinud, siis lisatakse saadud vorm, mis enamasti on lemma kujul, tagasi originaalteksti, õigele positsioonile. Seejärel luuakse eraldatud sõnadest taas ühine tervikteksti sisaldav sõne ning tulemus väljastatakse kasutajale. Lisas 2 on toodud ka programmi lähtekood, kus murdekorpuse meetod on realiseeritud.

2.3 Teisendusmeetod

Teisendusmeetod töötab regulaaravaldiste abil käitisi loodud reeglite põhjal. Antud töö raames on loodud reegleid kahele murdele, milleks on Saare ja Võru. Vastavaid teisenduste komplekte saab programmile hilisemalt ka juurde lisada, et toetatud murrete hulka tõsta. Teisendusmeetodi tööloogika seisneb murdesõnadele erinevate sobilike teisenduste tegemistes ning saadud tõlgetest õige välja filtreerimises. Taoline lahendus on pigem sobilik murretele, mis on kirjakeelele sõnakujude poolest lähedased ehk erinevad mõne tähe poolest.

Võrreldes klassikalise reeglipõhise masintõlkega [15] on antud lahendus primitiivsem, kuna ei arvesta lähtekeele ega siirdekeele süntaktilise struktuuri ega grammatiliste reeglitega. Oma

[3] http://www.murre.ut.ee/mkweb/

olemuse poolest ei hõlma see endas sõnastikke ega morfoloogilisi analüsaatoreid ja süntesaatoreid. Seega ei saa antud tõlkimismeetodit reeglipõhiseks masintõlkeks kvalifitseerida.

Algoritmi kirjeldus

Programmi sisendiks on tekst, millest filtreeritakse välja sõnad, mis ei kuulu kirjakeelde. Neid sõnu töödeldakse ükshaaval, seejuures rakendatakse igale neist teisendusi vastavalt määratud murdele. Näiteks Saare murdes olevad teisendused on:

- esimese *ö* asendamine *i*-ga,
- esimese *ä* asendamine *e*-ga,
- algusesse *h* lisamine,
- asendada *nd nud*-ga kombinatsioonidega,
- asendada *ö õ*-ga kombinatsioonidega,
- asendada *ei õi*-ga kombinatsioonidega,
- asendada *li lj*-ga kombinatsioonidega,
- asendada *ii üü*-ga kombinatsioonidega,
- asendada *ü i*-ga kombinatsioonidega.

Taoliste asenduste tegemiseks kasutakse regulaaravaldisi, et näiteks tuvastada, kas tegemist on sõna lõpuga või kas sõnas leidub otsitav täht või tähtede kombinatsioon. Näiteks „asenda *ö õ*-ga kombinatsioonidega“ reegel leiab etteantud sõnas kõik võimalikud kombinatsioonid, kuidas vastavaid sümboleid vahetada. Selleks minnakse rekursiivselt kahes harus sõnas üha sügavamale. Ühes harus tehakse teisendus, teises harus mitte. Nii jõutakse sõna lõpuni ning tagastatakse leitud vorm. Iga haru tagastab ühe võimaliku kombinatsiooni. Lisas 3 on toodud programmi „teisendus_meetod.py“ lähtekood ning seal paiknevad meetodid *teisendus_seto(sona)* ja *teisendus_saare(sona),* kus on võimalik täpsemalt võimalike teisendusreegleid näha.

Iga sõna puhul genereeritakse hulk sõnavorme, mida erinevad rakendatud teisendused väljastavad. Seejärel kontrollitakse, kas saadud hulgas leidub mõni tõlge, mis vastab eesti kirjakeelele. Juhul kui leidub, lisatakse see sõna tõlkeks ning märgistatakse „$“ sümbolitega. Juhul kui ükski teisendus ei suutnud aga sobivat vormi välja pakkuda, jäetakse algne ilma tõlketa sõna alles ning märgistatakse väljundis tähistustega („#“),.

Kui kõik filtreeritud murdesõnad on sel viisil läbitud, asendatakse algtekstis ära märgendatud tõlget vajavad sõnad teisenduste abil leitud vastavate tõlgetega. Seejärel tagastatakse kogu väljundtekst ühtse sõnena.

2.4 Statistilise masintõlke meetod

Statistiline masintõlge on masintõlke paradigma, kus tõlkeid genereeritakse statistilise mudeli alusel, mille parameetrid on hangitud mitme keele vahelisest paralleelkorpusest ning see on peamine alternatiiv reeglipõhisele masintõlkele [11]. Antud töö raames on katsetatud algelist masintõlke lahendust, mis suudaks murrete ja eesti keele paralleelkorpuste põhjal murdes olevat teksti tõlkida. Kasutatud on kõige algelisemat statistilise masintõlke keelemudeli algoritmi IBM Model 1, mis on mitmete tänapäeval kasutuses olevate statistiliste mudelite aluseks [2,9].

Algoritmi kirjeldus

Antud töös on kasutatud olemasolevat IBM Model 1 realisatsiooni, mille repositoorium[4] on avalikult GitHub keskkonnas üleval. Koodi on kohandatud, et see oleks võimeline murdetekste tõlkima.

Enne tõlkemeetodi kasutamist on tarvilik eeltöötlus juhul, kui keelemudel on eelnevalt loomata. Selleks tuleb esmalt paralleelkorpus sobivasse vormi viia. Antud töös on sobilikuks vormiks json-formaadis fail, kus on lähtekeele laused ja siirdekeele laused omavahel seotud. Et taolist faili koostada tuleb paralleelkorpuses olevad tekstid esmalt lausestada. Juhul kui mõlemas keeles olevaid lauseid on võrdne arv, viiakse laused üksteisega vastavusse ning saadud paaridest lisatakse json-formaadis fail, mis on mudeli treenimiseks sobiv vorming.

Treeningfail võtabki endale sisendiks paralleelkorpusest koostatud json-faili ning loeb selle siis Pythoni järjendi vormis, mis sisaldab sõnastik tüüpi objekte. Seejärel on võimalik loodud järjendi põhjal keelemudelit treenida.

Kasutatud programmis kogutakse kokku kõik erinevad sõnad, mis paralleelkorpuses on ning talletatakse need vastavalt keeltele kahte hulka. Seejärel initsialiseeritakse esialgsed tõlketõenäosused, kus igast eesti keelsest sõnast on ühtemoodi tõenäoline saada ükskõik milline

[4]http:// https://github.com/shawa/IBM-Model-1

võrukeelne sõna ehk iga lähtelause sõna joondatakse kõigi siirdelause sõnadega. Et saadud tõenäosusi parandada hakatakse jooksutama EM-algoritmi [2], mis käib läbi kõik korpuses leiduvad lausepaarid ning suurendab iga leitud joonduse korral tingliku tõenäosust, et vastava eestikeelse sõna puhul on tõlkeks just taoline võrukeelne sõna. Korpust käiakse läbi senikaua, kuni tõenäosused on koondunud ehk kahe tõlkenäosuste tabelite eukleidiline kaugus on väiksem kui konstandina määratud epsilon. Kui tõenäosuste sõnastik on koondunud, moodustatakse sellest mudel, kus iga lähtekeele sõnale määratakse tõlge vastavalt kõige suurema tingliku tõenäosusega sihtkeele sõnale. Valminud mudel tagastatakse tekstidokumendina. Saadud mudeli treenimine korpuse peal on vajalik vaid korra, kuna see on tekstifaili kujul salvestatud ning seda saab korduvalt anda ette tõlkeprogrammile sisendargumendina.

Tõlkeprogramm loeb sisse nii loodud mudelifaili kui ka sisendteksti. Sisendteksti eeltöötlus teeb antud juhul teksti lauseteks. Iga lause lastakse läbi tõlkefunktsiooni, mis tagastab lause tõlgitud versiooni vastavalt tõlkemudelist saadud tõenäolisemaile tõlkele. Kui kõik laused on töödeldud, luuakse neist uuesti terviklik sõne, mis väljastatakse. Statistilise meetodi lähtekood toodud ka antud töö lisas 4

3. Tulemused ja edasine areng

Bakalaureusetöö jaoks loodud kolme lahenduse tulemused on üpris erinevad. See on tingitud eelkõige sellest, et kõik meetodid lähenevad probleemile erineva nurga alt. Järgnevas peatükis tutvustatakse saadud tulemusi kolme lahenduse lõikes ning tuuakse välja nende tugevused ja puudujäägid.

3.1 Tulemuste analüüs

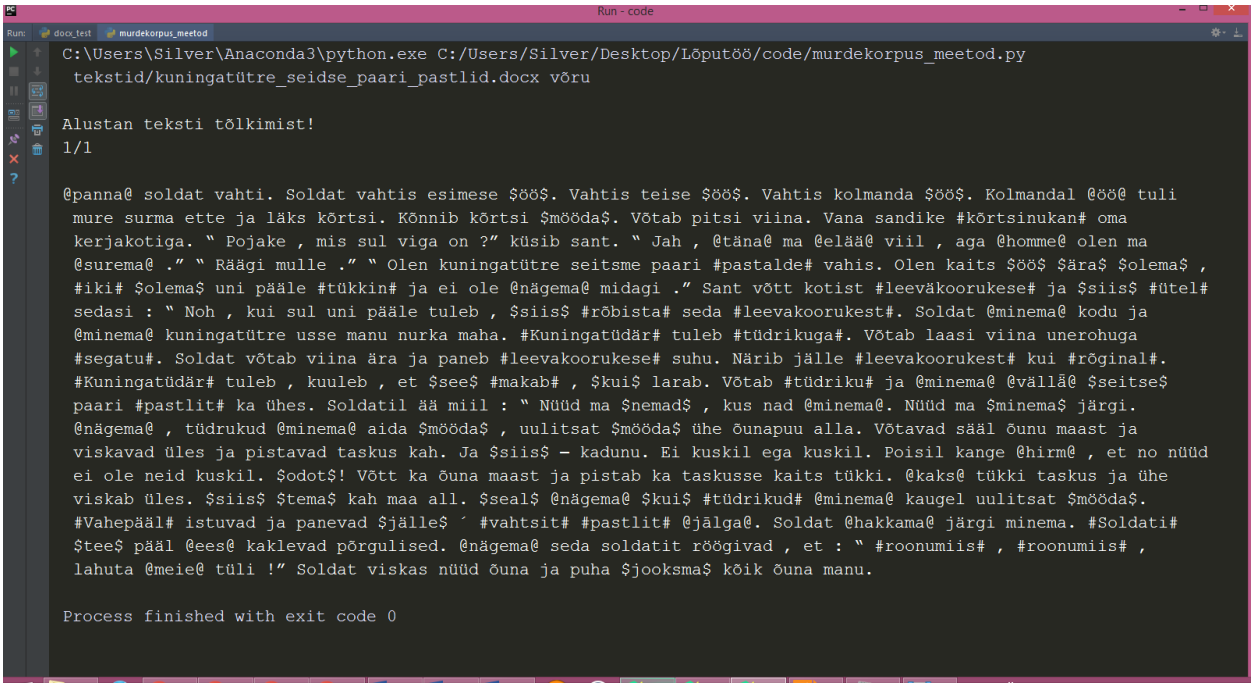
Kuna antud töö raames loodud lahendused on oma tõlkevõimekuse poolest erinevad, siis on nende võrdlemine raskendatud. Kõige enamate murretega suudab toime tulla murdekorpuse meetod. Teisendusmeetodil on töö valmimishetkel loodud reeglid Saare ja Võru murde tarbeks. Statistiline tõlkemeetod vajab oma tööks paralleelkorpust ning Eesti murretest eksisteerib see teadaolevalt vaid Võru keele jaoks ja ka see on üsna väikse mahuga, sisaldades endas palju ajakirjanduslikku ja reklaamteksti ning vähe ilukirjanduslikku, mis muinasjuttude tõlkimiseks sobilikum oleks [22]. Seega on tõlketulemuste võrdlemine kõigi kolme lahenduse puhul võimalik vaid Võru murde põhjal.

3.1.1 Võru murdes materjali tõlkimine kõigi kolme meetodiga

Testi aluseks oli osa Võru murdes muinasjutust „Kuningatütre seidse paari pastlit“, mille koguteksti võib leida lisast 5. Murdekorpuse programmi väljund on kujutatud joonisel 3, teisendusmeetodi väljund joonisel 4 ja statistilise meetodi väljund joonisel 5. Eeltöötlus, mis filtreeris tekstist välja mitte kirjakeelsed sõnad, tuvastas 75 tõlkimist vajavat juhtu.

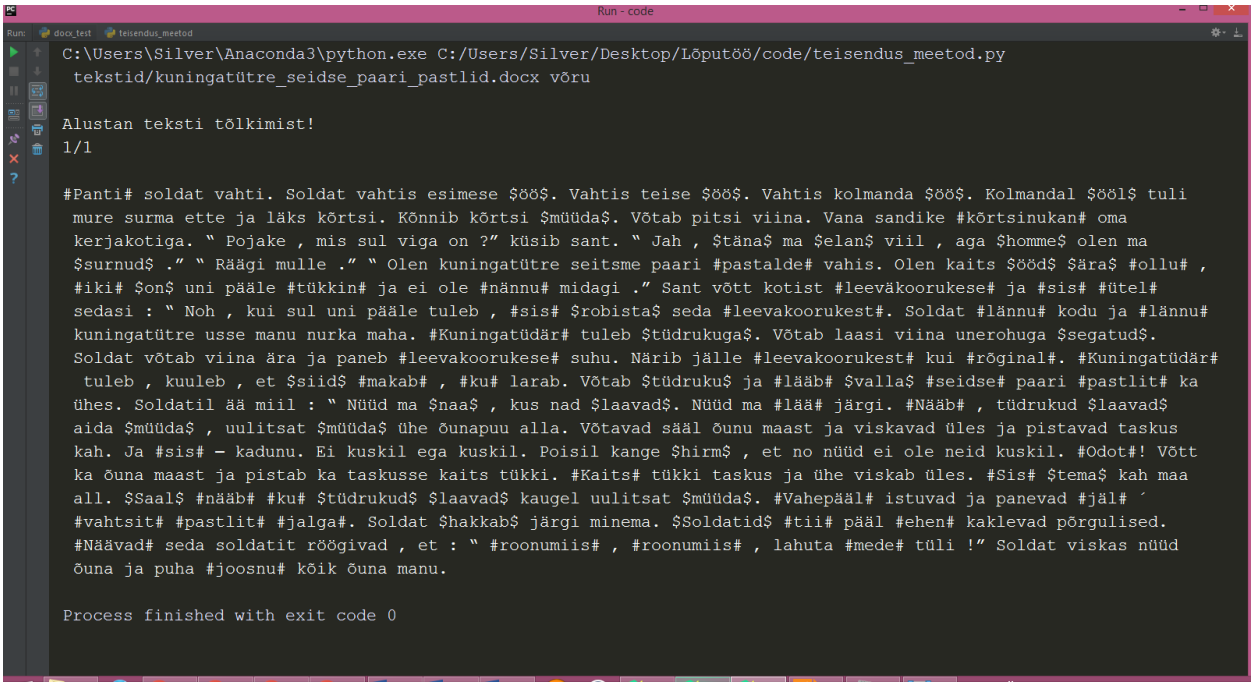
Murdekorpuse meetod (vt joonis 3) teostas ootuspäraselt kõige enam tõlkeid. 27 sõna said võrumurdelise tõlke, mis on 36%, 23 sõna, mis on 30,6% said mõnest teisest murdest potentsiaalselt õige tõlke ning 25 sõna, mis on 33,3% jäi programmil tõlkimata. Vähemalt mingisuguse tõlke sai endale seega 67,6% juhtudest. Enamik tõlketa jäänud sõnadest olid liitsõnad, nagu näiteks „leevakooruke“, „kuningatüdär“ või „roonumiis“, mille osasõnadele oskab murdekorpus tõlkeid pakkuda. Juhul kui programmi täiustada liitsõnade poolitajaga, mis tunneb murdekeelseid tekste, oleks võimalik taolised sõnad samuti ära tõlkida. Lisaks ei suutnud programm vastet leida käänetes ja pööretes olevatele vormidele, nagu näiteks „tüdrikud“. Sõnale „tüdrik“ on korpuses aga võrukeelne tõlge olemas.

Tagastatud teksti jäi sisse veel murdesõnu, mida algne kirjakeelsus kontrolli murdesõnadeks ei pidanud. Nendeks olid näiteks „usse“, mis antud lauses oleks pidanud tõlgitama sõnaks „ukse“ ja „miil“, mis oleks pidanud tõlgitama sõnaks „meel“.



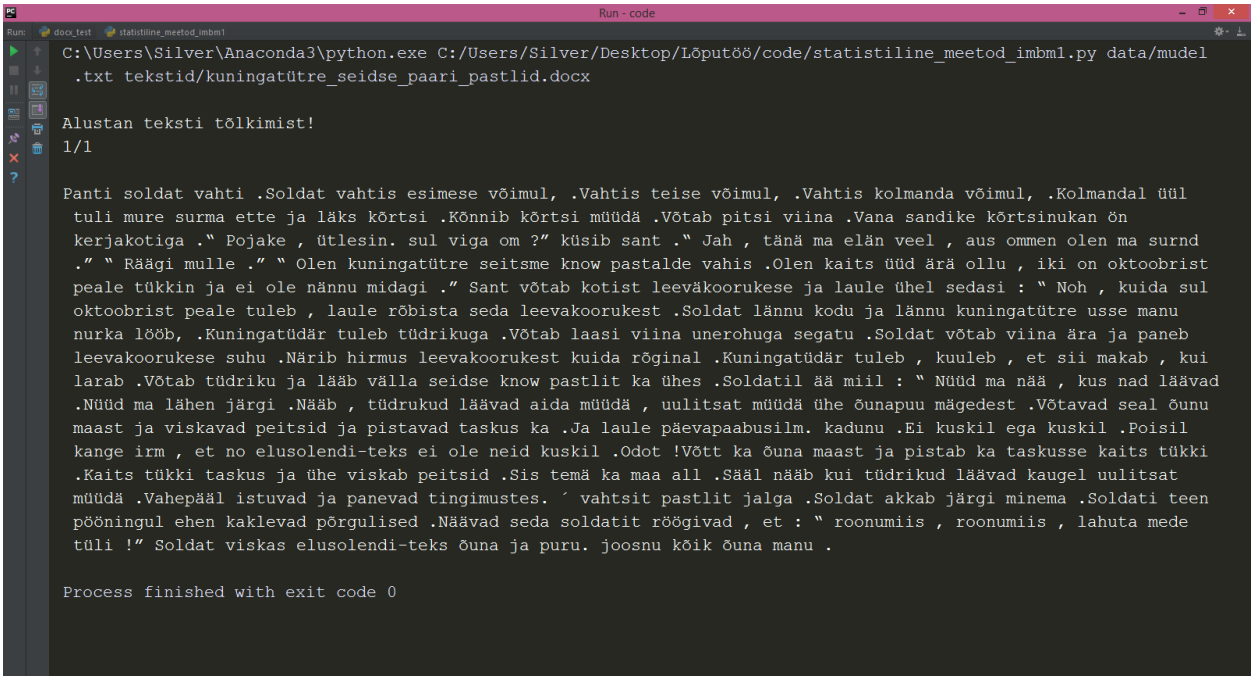
Joonis 3. Murdekorpuse meetodi väljund Võru murdes tekstiga

Teisendusmeetod suutis tõlketeisenduse teha 31-le sõnale, mis on ligikaudu 41,3% ehk tõlkimata jäi 44 potentsiaalset juhtu, mis on 58,6%. Küll aga pole antud meetodi puhul kõik leitud tõlked sobivad, sest teisenduste abil moodustatud vorm ei pruugi antud kontekstis olla korrektne tõlge. Näiteks osalause “Nääb, tüdrukud läävad aida müüdä…“ on tõlgitud „#Nääb# , tüdrukud $laavad$ aida $müüda$“, kus esimesele juhule pole leitud tõlget, teisele ja kolmandale juhule on leitud antud lausesse ebasobivad vormid. Korrektsed tõlked peaks olema „lähevad“ ja „mööda“.



Joonis 4. Teisendusmeetodi väljund Võru murdes tekstiga

Statistiline meetod suutis vaid loetud sõnadele korrektse tõlke välja pakkuda. Kokku tõlkis programm 70 sõna. Korrektsed neist olid aga koos tulemusega sõnapaarid: „viil“ – „veel“, „om“ – „on“, „võtt“ – „võtab“, „lää“ – „lähen“, „sääl“ – „seal“, „kah“ – „ka“ ja „ku“ – „kui“. Peamiseks põhjuseks kehva tulemuste taga on nii treeningkorpuse väiksus, mis antud testi ajal koosnes 1600 paralleelsest lausest kui ka lihtsakoeline mudeli treenimise algoritm.



Joonis 5. Statistilise masintõlke meetodi väljund Võru murdes tekstiga

Saadud tulemuste põhjal sai Võru murdes tekstiga kõige paremini hakkama murdekorpuse meetod, mis andis kirjakeelse tõlke täpselt 2/3 eelnevalt tuvastatud murdesõnale. Paremuselt teine oli teisendusmeetod. Selle tulemused jäid aga madalateks, kuna Võru keel erineb kirjakeelest enamus juhtudel rohkem kui mõne tähe jagu. Kuna aga hetkel olemas olevad reeglid väga põhjalike teisendusi sõnadega teha ei suuda, jääb antud meetodil taoliste murrakute jaoks võimekust puudu.

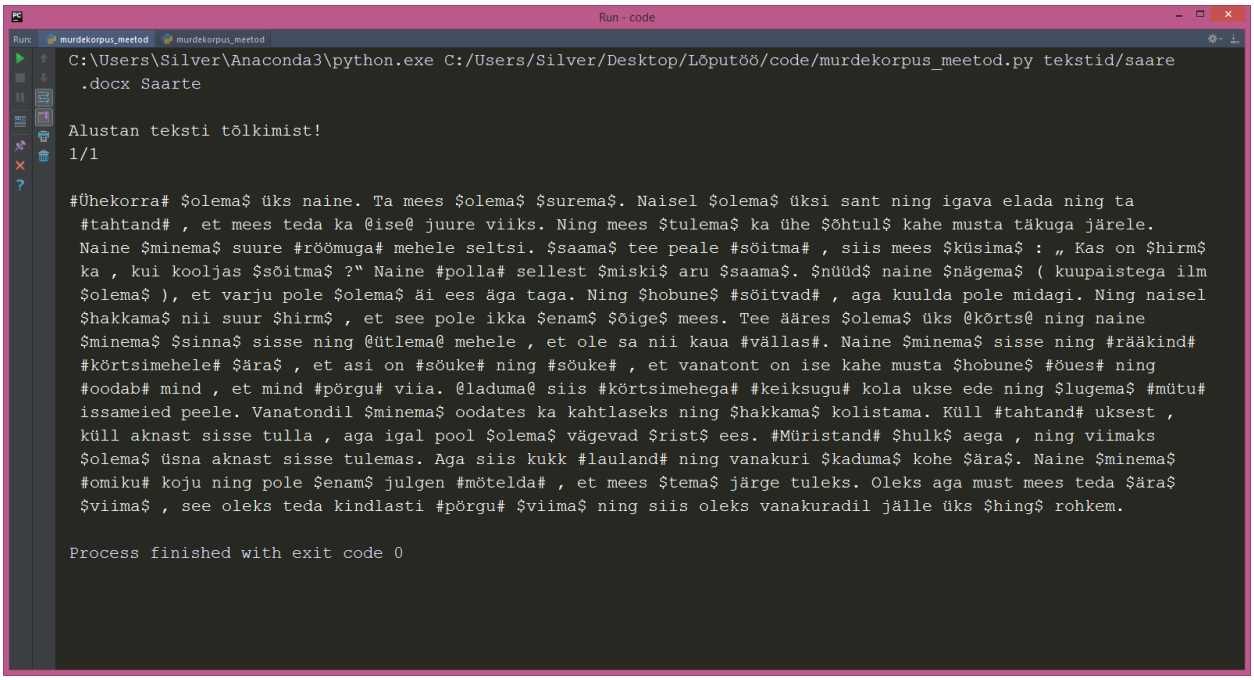
Statistiline meetod andis väga kesise tulemuse, olles suuteline vaid üksikuid sõnu korrektselt ümber tõlkima. Meetod vajaks õigemate tõlgete jaoks nii laiaulatuslikumat korpust kui ka keerukamat mudelit.

3.1.2 Saare murdes materjali tõlkimine murdekorpus- ja teisendusmeetodiga

Testi aluseks oli lühike Saare muinasjutt, mis on originaalvormis leitav lisast 6. Murdekorpuse meetodi väljund on kujutatud joonisel 6 ning teisendusmeetodi väljund joonisel 7. Tekstist leiti eeltöötlus faasis 71 murdesõna.

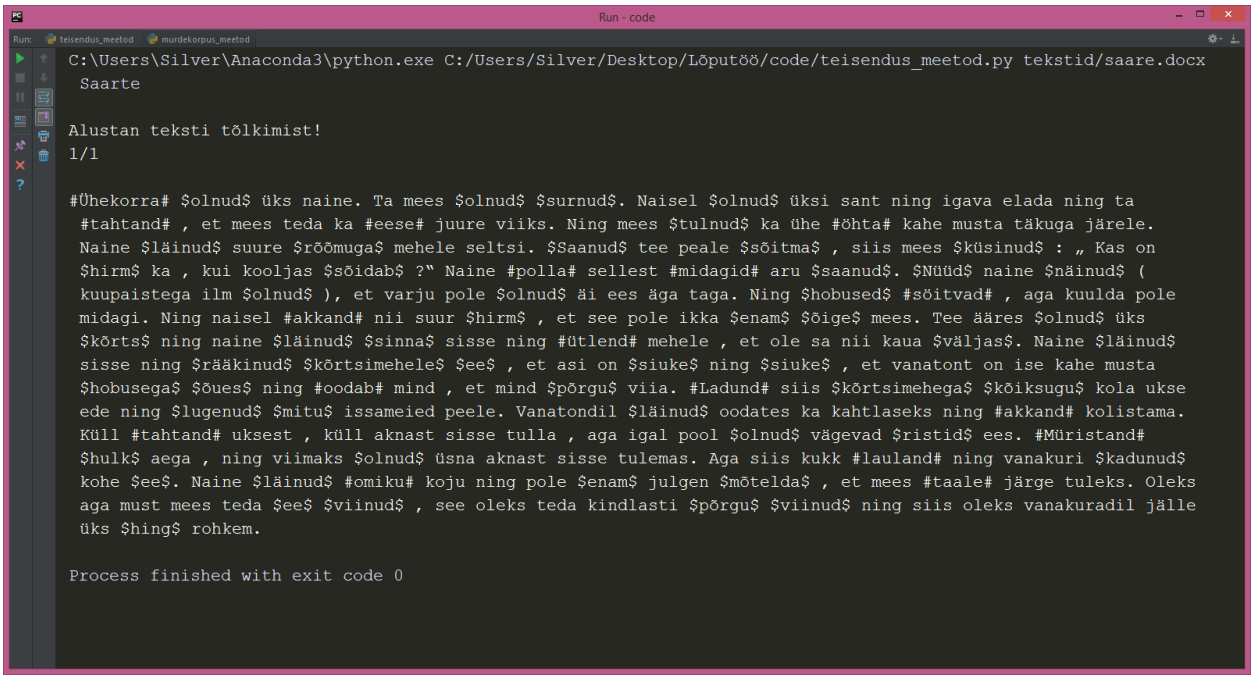
Murdekorpuse meetod suutis Saare murde järgi tõlkida 44 sõna ehk 62,0 % juhtudest. Mõne teise murde abil saadud potentsiaalsete tõlgete arv oli 4 ehk 5,6 % juhtudest. Seega kokku said tõlke

67,6 % murdesõnadest. Tõlketa jäi 23 sõna ehk 32,4 %. Nagu antud meetodile on omane, siis paljudele sõnadele leitakse tõlge, kuid väljundisse läheb pahatihti selle vale käändeline või pöördeline vorm.



Joonis 6. Murdekorpuse meetodi väljund Saare murdes tekstiga

Teisendusmeetod andis tõlke 54 sõnale, mis on 76,1 % kõikidest juhtudest ning jättis tõlketa 17 sõna, mis on 23,9 %. Antud meetodi puhul andis hea tulemuse see, et Saare murdes tekstides on kirjakeelega võrreldes palju väikseid erinevusi, mida on lihtne mõne tähe ümbermuutmise või sõna lõppu või algusesse lisamisega ühtlustada. Kuid ka Saare murde puhul genereeris meetod tõlkeid, mis antud lausesse ei sobi. Näiteks tõlgiti sõna „söuke“ sõnaks „sõuke“, kuid õige vorm oleks olnud „selline“. Paraku hetkel nii võimekad reeglid meetodil puuduvad.



Joonis 7. Teisendusmeetodi väljund Saare murdes tekstiga

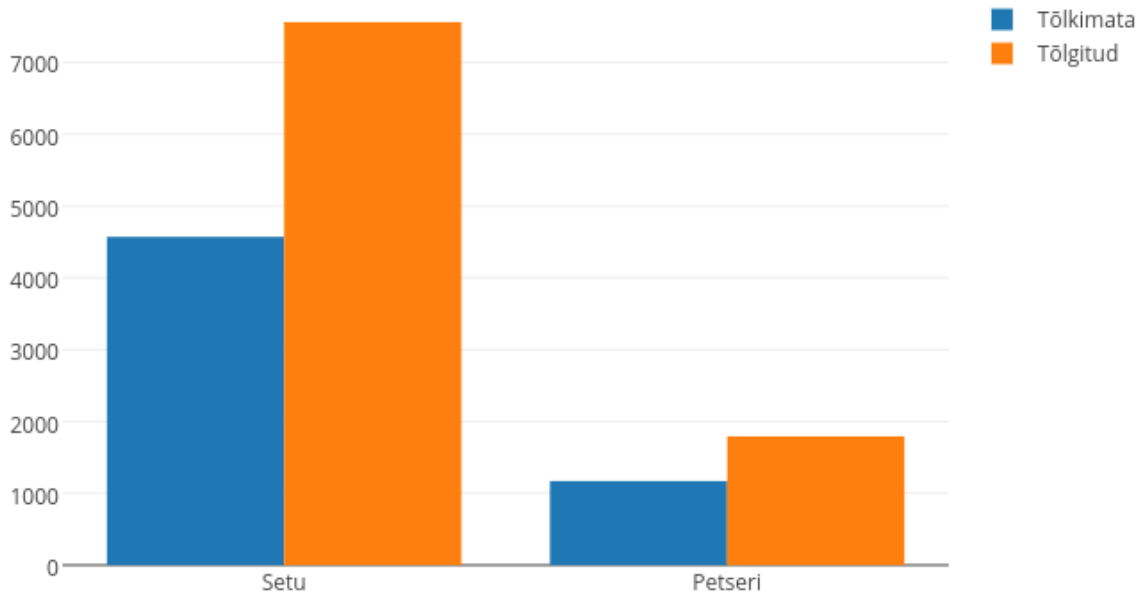
Antud teksti tõlkimisega tuli protsentuaalselt paremini toime teisendusmeetod, mis leidis tõlked 76,1% sõnadest, murdekorpuse 67,6% vastu. Küll aga on murdekorpuse meetodi väljund suurema tõenäosusega korrektsem, kuna see toetub oma tõlgete tegemisel murdekorpusele. Teisendusmeetodi genereeritud vormid on teksti loetavuse osas paremad, kui murdekorpuse tagastatud lemmad, kuna need säilitavad üldjuhul sõna käändelise või pöördelise vormi. Samas võivad genereeritud vormid olla antud lausesse sobimatud.

3.1.3 Murdekorpuse meetodi tulemused Kirjandusmuuseumi testmaterjalidega

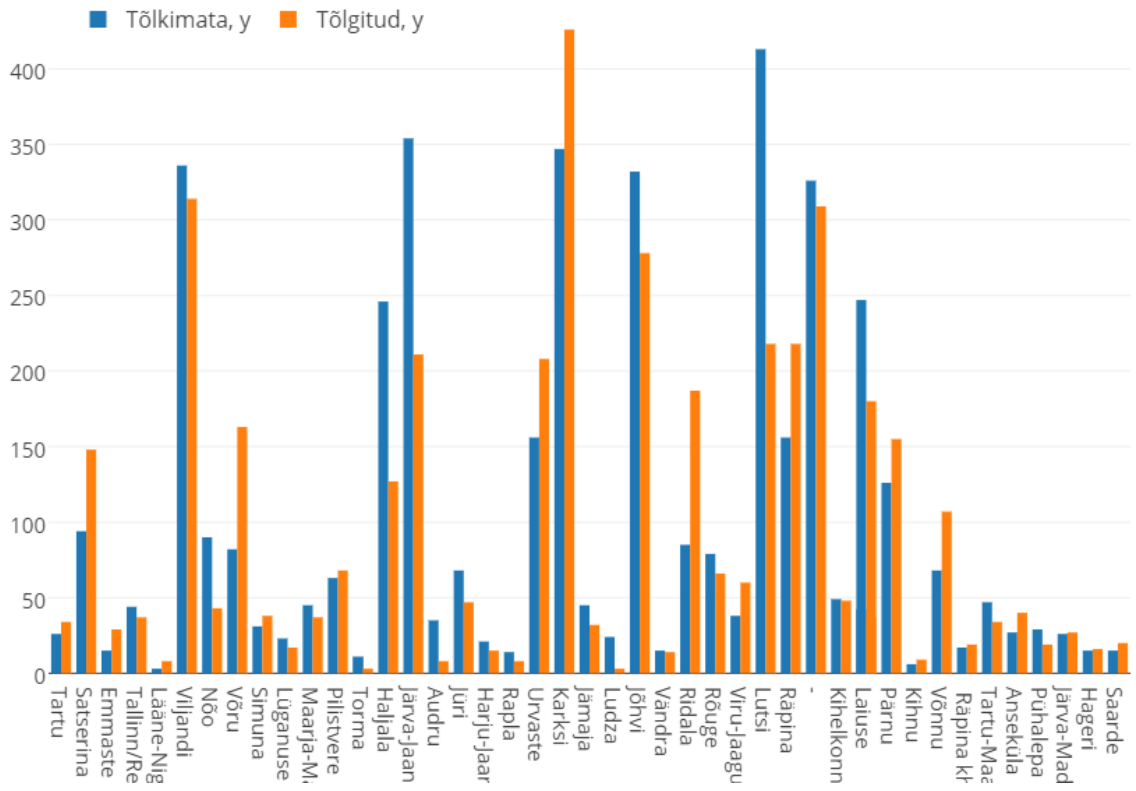
Antud lõputöö idee pärines Eesti Kirjandusmuuseumilt inimestelt, kes soovisid, et toimuks kogutud murdekeelsete muinasjuttude ühtsustamine kirjakeele baasil. Töö raames loodud rakendustest kvalifitseerub vastava ülesande tarbeks kõige paremini murdekorpuse meetod, sest sellel on võimekus tõlkida kõige enamates murretes tekste.

Testmaterjaliks on Kirjandusmuuseumi poolt väljapakutud muinasjuttude kogum, mis sisaldas endas 98 muinasjuttu. Joonisetel 8 ja 9 on kujutatud diagrammidena erinevatest asukohtadest saadud tulemusi. Ühes tulbas asetsevad sõnad, mis kvalifitseeriti tõlgitavaks ning millele leiti tõlge kas vastavast murdest või mõnest teisest murdest. Teises tulbas on sõnad, mis kvalifitseeriti

murdesõnadeks, kuid millele murdekorpus ühtki vastet ei suutnud pakkuda. Joonisel 8 on andmed Setu- ja Petserimaalt. Nendest paikadest pärit tekstides oli kõige enam murdelisi sõnu, kokku 17 435. Joonisel 9 on ülejäänud piirkondade andmed, kus murdesõnu tuvastati kokku 8426.



Joonis 8. Setu ja Petseri piirkondadest pärit muinasjuttude tõlgitud ja tõlkimata jäänud murdesõnad



Joonis 9. Piirkondadest pärit muinasjuttudest tõlgitud ja tõlkimata jäänud murdesõnad (va Setu ja Petseri)

Nagu joonisel 8 on näha, siis nii Setu kui Petseri kandist pärit tekstidele suudeti rohkem sõnu tõlkida (10 857 ehk 62,3%) kui tõlkimata jäta (6578 ehk 37,3%). Antud tekstid moodustavad Kirjandusmuuseumi muinasjuttude testitud andmekogust veidi rohkem kui veerandi, kuid sisaldavad sellegipoolest rohkem murdesõnu, kui ülejäänud piirkonnad kokku. Seega on tulemus positiivne.

Ülejäänud väiksemate piirkondade puhul on tulemused erinevad. Näiteks Lutsi, Järva-Jaani ja Laiuse kandist pärit tekstidel oli tõlkimata jäänud sõnade arv oluliselt suurem kui tõlgitute arv. Kokkuvõttes suudeti tekstidel, mis ei pärine Setu ega Petseri kandist tõlkida 4085 sõna ehk 48,5% ja tõlkimata jäi 4341 sõna ehk 51,5%. Nagu eelnevalt mainitud, siis antud meetod ei ole võimeline enamike murrete sõnade käändevorme tõlkima. Kuid kuna töödeldud tekstide arv enamike joonisel 9 kujutatud piirkondade puhul on üpris väike, ei saa nende alusel põhjapanevaid järeldusi teha.

3.2 Meetodite omapärad

Järgnev alapeatükk toob välja loodud lahenduste tugevad küljed ja puudused.

3.2.1 Murdekorpuse meetod

Murdekorpuse meetodi suurimaks eeliseks teiste ees oli laiem pagas erinevatest murretest, mis tähendas, et see sai hakkama kõige suurema arvu sõnade tõlkimisega üle kõigi testide. Samas oli selle meetodi suurimaks puuduseks võimekus toime tulla käändes ja pöördes olevate murdesõnadega, millele korpus ei suutnud adekvaatset tõlget pakkuda, kuna see sisaldab peamiselt sõnade lemmavorme.

Teiseks puuduseks on korrektsete tõlgete käänamine ja pööramine. Juhul, kui murdekorpus suudab käändes või pöördes olevale sõnale vastava tõlke leida, siis kirjakeelses tulemuses kajastub see ikkagi algvormis ehk lemmana, mitte sobivas käändes või pöördes nagu antud lause kontekstis oleks õige.

Mõlemat probleemi oleks võimalik lahendada morfoloogilise analüsaatori ja süntesaatori abil. Sel juhul tuvastatakse lähtetekstist murdesõna kääne või pööre ning pärast tõlkimist viiakse kirjakeelne sõna samasse vormi. See eeldab aga igale murdele omase morfoloogilise analüsaatori olemasolu. Alternatiivse lahendusena saaks kirjakeelset teksti ka tagantjärgi analüüsida ning tõlgitud sõnadele kõige tõenäolisemad vormid omistada ning seejärel morfoloogilise süntesaatoriga vormid vastavaks muuta. Selleks tuleks eelnevalt morfoloogiliselt analüüsitud murdetekste sisaldava korpuse peal valmis treenida vastav mudel. Taolisi korpuseid aga napib.

Lisaks saaks lahenduse efektiivust parandada, kui murdekorpuse veebiliidese päringute asemel hankida tõlkeid mõnest staatilisest struktuurist. Näiteks kõiki lemmasid ja nende tõlkeid koondavast failist. Hetkel on programmi töökindlus ja kiirus sõltuvuses kasutaja võrguühendusest ning selle kiirusest.

3.2.2 Teisendusmeetod

Teisendusmeetod võimaldab piisava hulga reeglite põhjal saada arvestatava tulemuse. Küll aga nõuab see suurel määral eelnevat käsitööd reeglite koostamisel. Reeglite lisandumisel suureneb ka valetõlgete ning potentsiaalsete konfliktide arv. Et taolist riski maandada, oleks üheks võimaluseks rakenduse kasutajal anda võimalus konfliktse tõlke puhul valida sobilik sõna käsitsi mitme

kirjakeelde sobiva tõlke hulgast. See aga tähendaks lahenduse täisautomaatsuse kadumist ning suurendaks käsitöö mahtu veelgi.

3.2.3 Statistiline masintõlke meetod

Statistiline meetod on antud lõputöös toodud meetoditest kindlasti kõige automaatsem, kuid selle korrektse toimimise eelduseks on mahuka paralleelkorpuse olemasolu. Selle töö raames sai kasutatud Võru keele instituudi paralleelkorpusest pärinevaid tekste, et treenida algeline statistiline tõlkemasin. Kuna aga loodud mudel on üsna madala intelligentsusastmega ehk mudeli treenimine toimub kõige lihtsakoelisemate eelduste põhjal ning selle treeningkorpus koosneb pooleteisttuhandest lausepaarist, siis pole saadud tõlked kvaliteetsed. Täpsust aitaks tõsta nii korpuse suurendamine kui ka rohkemate ilukirjanduslike tekstide lisandumine. Peale korpuse suurendamise, parandaks tulemusi ka keerukama treenimisalgoritmi loomine. Keerukam mudel on suuteline näiteks fraasituvastamisega toime tulema, seeläbi andes paremaid tulemusi, mis sobivad konteksti.

3.3 Arenemisvõimalused

Ükski töö raames valminud rakendustest ei ole täiuslik, sest need ei tagasta kasutajale täielikult kirjakeelde tõlgitud murdeteksti. Samas ei ole täiusliku tõlkega masintõlge ka antud töö eesmärk olnud. Küll aga on kõigil meetoditel külgi, mida saaks täiendada või teatud juhtudel ka ümber teha, et tulemust parandada.

Murdekorpuse meetod tuleks kindlasti staatilise struktuuri peale ümber kirjutada. See vähendaks programmi sõltuvust internetiühendusest ning kiirendaks oluliselt ka tõlkeprotsessi, kuna sõnale sobiliku tõlke leidmiseks poleks vaja teha eraldi veebipäringut, mis on kõige ajamahukamad operatsioonid antud programmi töös. Ka jääks ära CSV-failide käitlemine. Lisaks annaks lahenduse efektiivsusele juurde morfoloogiline analüsaator ja lemmatiseerija, mis suudaks murdesõnu algvormi viia. See tõstaks oluliselt tõenäosust, et murdekorpuses leiduks vastavale lemmale sobilik tõlge. Taoline analüsaator on arenduses Võru keele tarbeks, kuid on üsna ebatõenäoline, et kõigile Eesti murretele taolist tööriista valmistama hakatakse.

Teisendusmeetod vajaks oma efektiivsuse tõstmiseks rohkematele murretele vastavaid teisenduste kogumeid. Lisaks tuleks täiendada olemasolevaid Võru ja Saare keelte reeglistike. Kuna antud meetod töötab eelkõige kirjakeelele sarnasemate murretega, siis Võru murdele täielikult omaste

teisenduste tegemine on väga keeruline. Lisaks tuleks teisendused omavahel kaskaadis tööle panna ehk ühe teisenduse tulemust kombineerida teisega, mis moodustaks veelgi enam võimalike sõnavorme, mille kirjakeelsust oleks võimalik testida. Ka saaks meetodi ümber teha peatükis 1.3 kirjeldatud lõplike olekumasinate teisendustele. Kirjakeelele sarnasemate murrete puhul saaks rakendada ka mainitud automaatset reeglite tuvastamist [6], seeläbi kaotades suure osa käsitööst.

Kui analüüsida murdekorpuse ja teisendusmeetodi lahenduste tulemusi, siis võib neid uurides tähele panna, et kõige kvaliteetsema tõlke saaks mõlema meetodi tagastatud korrektseid tõlkeid kombineerides. Selleks tuleks muuta mõlema meetodi struktuuri ning konstrueerida algoritm, mis suudaks otsustada kumma lahenduse tõlge on sobilikum. Kuigi lihtsama lahenduse saaks juba olemasolevate meetoditega implementeerida, kasutades näiteks teisendusmeetodit ainult sõnade puhul, mida murdekorpus ära ei suutnud tõlkida.

Statistiline meetod vajaks kindlasti ümberkirjutamist, sest antud töö tarbeks kasutatud lahendus ei ole piisav, et luua mudeleid korpuste peal, mis on suuremad kui mõnituhat lauset. Lisaks on IBM Model 1 liiga algeline treenimismudel, mida tuleks kindlasti edasi arendada kõrgema taseme statistiliste mudeliteni. H.-J. Kaalepi ja M. Koidu hinnangul [9] on soome-ugri keelte tarbeks taolised mudelid puudulikud, kuna need ei tule toime rikkaliku morfoloogia ja vaba sõnavaraga keeltega. Nii eesti keel kui ka Võru murre kuuluvad antud liigituse alla. Seega on parema algoritmi loomine ülimalt oluline. Ka on oluline laiaulatuslike paralleelkorpuste olemasolu. See on aga probleem, mida pole kerge ilma suure käsitööta lahendada, kuna vastavaid korpuseid erinevatest Eesti murdetekstidest ja nende kirjakeelsetest tõlgetest lihtsalt ei eksisteeri.

4. Kokkuvõte

Antud lõputöö eesmärk oli luua lahendus, mis võimaldaks erinevates Eesti murretes tekste ühtlustada, et neid oleks võimalik kergemini tervikuna käsitleda ning analüüsida. Selleks prooviti kolme töö raames loodud meetodit, mille abil teisendada murdetekste eesti kirjakeelde. Nendeks olid murdekorpuse meetod, teisendusmeetod ja statistiline masintõlkemeetod. Kõik kolm meetodi olid erineva võimekusega. Näiteks murdekorpuse meetod suutis tõlkida pea kõigis murretes olevaid tekste, kuid statistiline meetod toimis vaid Võru murdes tekstidega.

Saadud testitulemuste põhjal oli kolme lahenduse võrdluses kõige edukam murdekorpuse meetod, mis tagastas ühise Võru murdes testteksti töötlemisel kõige enam tõlgitud sõnu. Samas oli saarte murdes teksti töötlemisel kõige edukam teisendusmeetod. Statistiline meetod vajab hulganisti edasiarendust, et see oleks võimeline treenima kvaliteetsemaid mudeleid. Kuid pelgalt keerukuse tõstmisest ei piisa, sest mudelite treenimise eelduseks on ka korraliku korpuse olemasolu. Tänu murdekorpuse meetodi universaalsusele sai seda testida ka mõnede Eesti Kirjandusmuuseumi muinasjuttude peal ning saadud tulemused olid lootustandvad. Kõige rohkem murdesõnu omanud murrete puhul tõlgiti ära tunduvalt rohkem sõnu, kui jäeti tõlkimata.

Siiski on kõigil kolmel meetodil arenemisruumi, et saadud tulemusi parandada ning tõlgete kvaliteeti tõsta. Lisaks on mitmeid võimalusi, kuidas tulevikus programmide tööd optimeerida. Paljuski on aga Eesti murrete masintõlge sõltuv sellest, kas neile luuakse tulevikus keeletehnoloogilisi tööriistu juurde või kas tegeletakse murdetekstide korpuste loomisega. Küll aga on vähemalt murdekorpuse meetod ja teisendusmeetod võimelised vähemalt osaliselt murdes tekste ümber teisendama juba töö raames valminud osas.

Viidatud kirjandus

[1] Bolotnikov K. The Many Dialects of China. Asia Society. http://translation-blog.trustedtranslations.com/india-the-country-with-more-than-2000-dialects-2011-10-18.html (10.05.2017)

[2] Collins M. "Statistical Machine Translation: IBM Models 1 and 2", University of Columbia. http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/ibm12.pdf (01.05.2017)

[3] Erelt M., Erelt T., Ross K. Eesti Keele Käsiraamat, Eesti Keele Instituut. 1997 https://www.eki.ee/books/ekk09/index.php?p=1&p1=2 (12.03.2017)

[4] Estnltk: Pythoni teegid eestikeelsete vabatekstide lihtsamaks töötlemiseks. Eesti Keeletehnoloogia Riikilik Programm (2011 – 2017). https://www.keeletehnoloogia.ee/et/ekt-projektid/estnltk-pythoni-teegid-eestikeelsete-vabatektside-lihtsamaks-tootlemiseks (26.04.2017)

[5] e-Teatmik: IT ja sidetehnika seletav sõnaraamat. http://www.vallaste.ee/index.htm

[6] Hulden M., Alegria I., Etxeberria I., Maritxalar M. Learning word-level dialectal variation as phonological replacement rules using a limited parallel corpus. First Workshop on Algorithms and Resources for Modelling of Dialects and Language Varieties. 2011

[7] Hulden M. Regular expressions and predicate logic in finite-state language processing. *Frontiers in Artificial Intelligence and Applications*, 191, 2009

[8] Jurafsky D., Martin J.H. Regular expressions, Text Normalization, Edit Distance. Speech and Language Processing, 2016, p 11-17

[9] Kaalep H.-J., Koit M. Kuidas masin tõlgib. *Keel ja Kirjandus*, 2010, nr 10, lk 730–735

[10] Kask A. Eesti murde ja kirjakeel. Tallinn „Valgus". 1984. lk 5-28

[11] Koehn P. "Statistical Machine Translation". Cambridge University Press, 2009

[12] Kommel K. Eesti murded rahvaloenduse andmetel. 2013. lk 4-6. www.stat.ee/dokumendid/71338 (12.03.2017)

[13] Marimuhtu K., Devi S. L. Automatic Conversion of Dialectal Tamil Text to Standard Written Tamil Text using FSTs.. MIT Campus of Ann University..2014. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.675.5027&rep=rep1&type=pdf (10.05.2017)

[14] Murdekorpus. Tartu Ülikooli eesti ja üldkeeleteaduse instituut. http://www.keel.ut.ee/et/keelekogud/murdekorpus (26.04.2017)

[15] Osborne M. History and Rule-based systems. University of Edinburgh https://web.archive.org/web/20120306014535/http://www.inf.ed.ac.uk/teaching/courses/mt/lectures/history.pdf (30.04.2017)

[16] Pajusalu K., Hennoste T., Niit E., Pällm P., Viikberg J. Eesti murded ja kohanimed. Eesti Keele Sihtasutus. 2009.

[17] Raag R. Talurahva keelest riigikeeleks. Atlex. 2008, lk 14-18.

[18] Sawaf H. Arabic Dialect Handling in Hybrid Machine Transaltion. 2010 https://amta2010.amtaweb.org/AMTA/papers/2-05-Sawaf.pdf (10.05.2017)

[19] Scott J. India, the Country with More Than 2,000 Dialects. Trusted Translations. http://translation-blog.trustedtranslations.com/india-the-country-with-more-than-2000-dialects-2011-10-18.html (10.05.2017)

[20] Söit, söit, söit Sörve poole – Saaremaa folkloori veebikogumik. http://saaremaa.folklore.ee/wordpress/?cat=64 (08.05.2017)

[21] Stephenson S. Search Trough Sound: Finding Phrases in Audio http://blog.deepgram.com/search-through-sound-finding-phrases-in-audio/ (04.05.2017)

[22] Võru keele paralleelkorpus. http://voro.aader.org/wi.py?z=vat (04.05.2017)

[23] Teksti morfoloogiline analüsaator. Eesti Keele Instituut e-keelenõu. http://kn.eki.ee/tool/?m=morfoloogia (10.03.2017)

Lisad

Lisa 1. Programmide käivitus- ja kasutusjuhend

Käivitamiseks on mõeldud programmid „murdekoprus_meetod.py“, „teisendus_meetod.py“ ja „statistiline_meetod_ibm1.py“

„murdekoprus_meetod.py“ ja „teisendus_meetod.py“ käivitamine käsurealt

Hoiatus – programm vajab võrguühendust, ülikooli võrgus (eduraom) on päringute kiirus piiratud, seetõttu on soovitatav testimisperioodiks ühenduda ut-public võrku.

1. Käivitada ilma argumentideta. Sel juhul on võimalik otse käsuviipa tõlgitavat teksti sisestada

2. Käsurea argumentidega. Sel juhul on võimalik tekstifaile (.txt ja .docx) sisendiks anda. Argumendid anda kujul "faili_asukoht murde_nimetus"

3. Käsurea argumendina spetsiaalne muinasjuttude kogumik (faili nimi peab lõppema \"mjp.docx\"). Sel juhul piisab vaid ühest argumendist, milleks on faili asukoht

„statistiline_meetod_ibm1.py“ käivitamine käsurealt

1. Käivitada mudelifaili ja tõlgitava lausega, kujul "mudelifaili_asukoht "tõlgitav_lause" (jutumärkides ehk ühtse sõnena)".

2. Käivitada mudelifaili ja sisendfailiga. Sel juhul on võimalik tekstifaile (.txt ja .docx) sisendiks anda. Argumendid anda kujul "mudelfaili_asukoht sisendfaili_asukoht".

3. Käivitada mudelifaili ja spetsiaalse muinasjuttude kogumikuga (faili nimi peab lõppema "mjp.docx"). Argumendid anda kujul "mudelfaili_asukoht sisendfaili_asukoht“

Iga meetodi käivitamisel kuvatakse samuti vastavad instruktsioonid, kuidas antud meetodeit kasutada.

Veelgi detailsem juhend asub failis „readme.txt“.

Miinimum nõuded programmide jooksutamiseks:

1. Python 3.5.*
   Kättesaadav: https://www.python.org/downloads/

2. EstNLTK 1.4.1 teek

   installeerimis juhend: https://estnltk.github.io/estnltk/1.4.1/tutorials/installation.html

3. Python-docx teek 0.8.6

   installeerimis juhend: https://python-docx.readthedocs.io/en/latest/user/install.html

4. Clize teek 4.0.1

   Installeerimis juhend: http://clize.readthedocs.io/en/stable/basics.html

Lisa 2. murdekorpus_meetod.py lähtekood

```
1.  import urllib.request
2.  import csv
3.  import requests
4.  from collections import defaultdict
5.  import sys
6.  import docx_test
7.  import eeltootlus
8.  import os.path
9.
10. # Tõlkemeetod. Saab sisendiks teksti ja vastava murde.
11. # Üritatakse leida murdele vastav tõlge.
12. # Juhul kui ei leita võtekse tõlge mõnest teisest murdest
13. def tolgi_tekst(tekst, murre):
14.     murde_sonad_eemaldatud, ainult_murde_sonad = eeltootlus.eemalda_kirjakeelsed_sonad(
    tolke_tekst_list)
15.     global tolgitava_jutuarv
16.     global jutude_koguarv
17.
18.     print()
19.     print("Alustan teksti tõlkimist!")
20.     print(str(tolgitava_jutuarv) + "/" + str(jutude_koguarv))
21.     print()
22.
23.     # Tuvastatud murdesõnadele murdekorpusest vaste leidmine
24.     for word in ainult_murde_sonad:
25.         test_word = word
26.
27.         # Päring korpuse veebiliidsele
28.         url = "http://www.murre.ut.ee/mkweb/ekspordi.csv.php?lemma=&liik=&tahendus=&son
    e=" + urllib.parse.quote(
29.             test_word) + "&vorm=&kontekst=&keel=&murre=&murrak=&aasta=&vanus=&sugu=&bos
    s="
30.         with requests.Session() as session:
31.             download = requests.get(url, stream=True)
32.             decoded_content = download.content.decode('utf-8')
33.
34.             # CSV faili parsimine
35.             cr = csv.reader(decoded_content.splitlines(), delimiter=',')
36.             my_list = list(cr)
37.             # Juhul kui tõlge puudub märgistatakse sõna "#" sümbolitega
38.             if len(my_list) == 1:
39.                 lopp_tolge = "#" + test_word + "#"
40.             else:
41.                 potent_oige_murre_tolked = defaultdict(int)
42.                 potent_vale_murre_tolked = defaultdict(int)
43.                 for tolke_valik in reversed(my_list[1:]):
44.                     splited_row = tolke_valik[0].split('\t')
45.                     tolge_1 = splited_row[1]
46.                     # Juht, kui tõlge on, aga murre puudub CVS tabelist
47.                     if len(splited_row) < 7:
48.                         murre_cvs = "-"
49.                     else:
50.                         murre_cvs = splited_row[6]
51.                     if murre_cvs.lower() == murre.lower():
52.                         potent_oige_murre_tolked[tolge_1] += 1
53.                     potent_vale_murre_tolked[tolge_1] += 1
54.
```

```
                # Potentsiaalsetest tõlgetest leida kõige rohkem esinev. Korrektse murde korral märgistada "$" sümbolitega, vale murde korral "@" sümbolitega
                if (len(potent_oige_murre_tolked) > 0):
                    lopp_tolge = "$" + max(potent_oige_murre_tolked, key=potent_oige_murre_tolked.get) + "$"
                else:
                    lopp_tolge = "@" + max(potent_vale_murre_tolked, key=potent_vale_murre_tolked.get) + "@"

            murde_sonad_eemaldatud[murde_sonad_eemaldatud.index("#")] = lopp_tolge

    # Tulemuse kokkupanek ühtseks sõneks
    lopptulemus_tekst = ""
    for i in range(len(murde_sonad_eemaldatud)):

        try:
            if murde_sonad_eemaldatud[i + 1] in ".!?":
                sona_eraldaja = ""
            else:
                sona_eraldaja = " "
        except:
            sona_eraldaja = ""

        lopptulemus_tekst += murde_sonad_eemaldatud[i] + sona_eraldaja


    return lopptulemus_tekst









sys.stdout.write("\033[1;36m")
print("Programmi kasutamiseks on kolm võimalust.\n"
      "1. Käivitada ilma argumentideta. Sel juhul on võimalik otse käsuviipa tõlgitavat teksti sisestada\n"
      "2. Käsurea argumentidega. Sel juhul on võimalik tekstifaile (.txt ja .docx) sisendiks anda. Argumendid anda kujul \"faili_asukoht murde_nimetus\"\n"
      "3. Käsurea argumnedina spetsiaalne muinasjutude kogumik (faili nimi peab lõppema \"mjp.docx\". Sel juhul piisab vaid ühest argumnedist, milleks on faili asukoht")
sys.stdout.write("\033[0;0m")

tulemused = {}

sonastik = set()

tolke_tekst_list=[]
murre = ""

      jutude_koguarv = 1
      tolgitava_jutuarv = 1

      if len(sys.argv)>2:

          # Tavaliste tekstifailide tõlkimine
          if os.path.isfile(sys.argv[1]):
              jutude_koguarv = 1
              tolgitava_jutuarv = 1
```

```
                if len(sys.argv) > 2:
                    file = sys.argv[1]
                    murre = sys.argv[2]
                    tolke_tekst_list = eeltootlus.get_file_content(file)
                tolge = tolgi_tekst(tolke_tekst_list, murre)
                print(tolge)
            else:
                sys.stdout.write("\033[1;31m")
                print()
                print("Esimene argument pole fail! Kontrollige, sisestatud faili path ol
    eks korrektne")
                sys.stdout.write("\033[0;0m")

        elif len(sys.argv)==2:
            # Kontoll kas üritatakse tõlikda Kirjandusmuuseumi testdokumente
            if "mjp.docx" in sys.argv[1] and os.path.isfile(sys.argv[1]):

                # Eraldi funktsioon, mis oskab vastavalt vormistatud .docx faile töödeld
    a
                kandid, jutud = docx_test.tagasta_sobiv_vorming(sys.argv[1])

                # Statistikasõnastiku initsailiseerimine
                for kant in set(kandid):
                    tulemused[kant] = (0, 0)

                jutude_koguarv = len(jutud)
                tolgitava_jutuarv = 0
                for i in range(len(jutud)):
                    tolgitava_jutuarv = i + 1
                    murre = kandid[i]
                    jutt = ""
                    for ptk in jutud[i]:
                        jutt += ptk
                    tolke_tekst_list = eeltootlus.sonasta(jutt)

                    tolge = tolgi_tekst(tolke_tekst_list, murre)
                    print(tolge)
                    print("Päritolu: ", murre)

                    # Statistika kogumine
                    endine_tulemus = tulemused[murre]
                    uus_tulemus = (endine_tulemus[0] + tolge.count("#") / 2,
                                   endine_tulemus[1] + tolge.count("@") / 2 + tolge.coun
    t("$") / 2)
                    tulemused[murre] = uus_tulemus
                print(tulemused)
            else:
                print()
                sys.stdout.write("\033[1;31m")
                print("Sisestatud ainult üks argument!\n"
                      "Argumendid anda kujul \"faili_asukoht murde_nimetus\"")

        else:
            print("Sisesta tõligtav tekst: ")
            input_text = input("")
            print("Sisesta teksti murre (Võru, Saarte, Lääne, Kesk, Ida, Mulgi, Ranna, S
    etu, Tartu, Alutaguse, läänеliiva, läänevadja, idaliiva, idavasja, Ira...)")
            murre = input("")
            tolke_tekst_list = eeltootlus.sonasta(input_text)
            tolge = tolgi_tekst(tolke_tekst_list, murre)
            print(tolge)
```

Lisa 3. teisendus_meetod.py lähtekood

```
import re
import sys
import docx_test
import eeltootlus
import os.path


#Teisendus funktsioonid
######################

# Setu ja Võru murde teisendused
def teisendus_seto(sona):

    return [sona,
          asenda_kõik("q","",sona),
          eemalda_lõpust("ä",sona),
          eemalda_lõpust("b", sona),
          eemalda_lõpust("n",lisa_algusesse("h", sona)),
          asenda_kõik("ä","a",sona),
          asenda_kõik("oi","oe",sona),
          asenda_kõik("ai", "ae", sona),
          lisa_lõppu("d",sona),
          asenda_kõik("õ","o",sona)] + asenda_kõik_kombinatsioonidega("m","n",sona) \
           + asenda_kõik_kombinatsioonidega("õ","u",sona)\
           + asenda_kõik_kombinatsioonidega("õh","a",sona) \
           + asenda_kõik_kombinatsioonidega("ü", "ö", sona) \
           + asenda_kõik_kombinatsioonidega("t", "d", sona) \
           + asenda_kõik_kombinatsioonidega("ri", "ru", sona) \
           + asenda_kõik_kombinatsioonidega("ä", "a", sona) \
           + asenda_kõik_kombinatsioonidega("nd", "nud", sona)



# Saarte murde teisendused
def teisendus_saare(sona):

    kombid_1 = asenda_kõik_kombinatsioonidega("ö", "õ", sona)
    kombid_2 = asenda_kõik_kombinatsioonidega("ei", "õi", sona)
    kombid_3 = asenda_kõik_kombinatsioonidega("ll", "lj", sona)
    kombid_4 = asenda_kõik_kombinatsioonidega("ii", "üü", sona)
    kombid_5 = asenda_kõik_kombinatsioonidega("ü", "i", sona)
    kombid_6 = asenda_kõik_kombinatsioonidega("nd", "nud", sona)

    return [sona,
          asenda_esimene("ö", "i", sona),
          asenda_esimene("ä", "e", sona),
          lisa_algusesse("h", sona)
          ]+ kombid_1 + kombid_2 + kombid_3 + kombid_4 + kombid_5 + kombid_6


# Tähiste asendus teisendus. Leiab kõik võimalikud asendus kombinatsioonid.
def asenda_kõik_kombinatsioonidega(asendatav,asendaja,sona):
    asendatav_arv = sona.count(asendatav)

    indeksid=[]
```

```
58.     if asendatav_arv == 0:
59.         return indeksid
60.     else:
61.         indeksid=indeksi_tuvastus(sona,asendatav,len(sona))
62.         kombid = kombinatsioonide_genereerimine(indeksid,sona,asendatav,asendaja)
63.
64.         return kombid
65.
66. # Rekursiivne funktsioon, mis ühes harus asnedab sisendsõnes leiduva tähise teises mitt
    e, seeläbi genereerides kõik võimalikud kombinatsioonid
67. def kombinatsioonide_genereerimine(indeksid,sona,asendatav,asendaja):
68.     if len(indeksid)==0:
69.         return [sona]
70.     else:
71.         indeks = indeksid[0]
72.         uus_sona=aseenda_positioonil(asendatav,asendaja,indeks,sona)
73.         muudetud = kombinatsioonide_genereerimine(indeksid[1:],uus_sona,asendatav,asend
    aja)
74.         muutmata = kombinatsioonide_genereerimine(indeksid[1:],sona,asendatav,asendaja)
75.
76.         return muudetud + muutmata
77.
78. # Leiab sõnest kõik otsitava tähise indeksid ning tagastab need järjendina
79. def indeksi_tuvastus(sona,asendatav,sonalen):
80.     indeks = sona.index(asendatav)
81.     oige_indeks = sonalen - len(sona) + indeks
82.     if sona.count(asendatav)==1:
83.         li = [oige_indeks]
84.         return li
85.     else:
86.         li=indeksi_tuvastus(sona[(indeks+1):],asendatav,sonalen)
87.         li += [oige_indeks]
88.         return li
89.
90.
91.
92. def asenda_kõik(asendatav,asendaja,sona):
93.     return re.sub(asendatav, asendaja, sona)
94.
95.
96. def asenda_esimene(asendatav,asendaja, sona):
97.     return re.sub(asendatav,asendaja,sona,1)
98.
99.
100.        def asenda_teatud_arv(asendatav, asendaja, asenduste_arv, sona):
101.            return re.sub(asendatav, asendaja, sona, asenduste_arv)
102.
103.        def aseenda_positioonil(asendatav, asendaja, algus_indeks, sona):
104.            osa = sona[algus_indeks:]
105.            esimene_pool=sona[:algus_indeks]
106.            if re.search(asendatav,sona):
107.                uus_osa = re.sub(asendatav,asendaja,osa,1)
108.            else:
109.                uus_osa = osa
110.
111.            return esimene_pool + uus_osa
112.
113.        def lisa_juurde(lisatav, algus_indeks, sona):
114.            return sona[:algus_indeks] + lisatav + sona[algus_indeks:]
115.
```

```
116.        def lisa_lõppu(lisatav, sona):
117.            return sona + lisatav
118.
119.        def lisa_algusesse(lisatav, sona):
120.            return  lisatav + sona
121.
122.        def eemalda_lõpust(eemaldatav, sona):
123.
124.            if (re.fullmatch("\w*" + eemaldatav + "$", sona)):
125.                uus_sona = re.sub(eemaldatav, "", sona[::-1], 1)
126.                uus_sona = uus_sona[::-1]
127.            else:
128.                uus_sona=sona
129.
130.            return uus_sona
131.
132.        ######################
133.
134.
135.
136.        # Tõlkemeetod. Saab sisendiks teksti ja vastava murde ning käivitab selle järgi
    teisendusfunktsioonid
137.        def tolgi_tekst(tekst,murre):
138.            # Eraldatakse sõnad, mis on murdelised.
139.            murde_sonad_eemaldatud, ainult_murde_sonad = eeltootlus.eemalda_kirjakeelsed
    _sonad(tolke_tekst_list)
140.            tolked = {}
141.
142.            global tolgitava_jutuarv
143.            global jutude_koguarv
144.
145.            print()
146.            print("Alustan teksti tõlkimist!")
147.            print(str(tolgitava_jutuarv) + "/" + str(jutude_koguarv))
148.            print()
149.
150.            if murre.lower() == "seto" or murre.lower() == "setu" or murre.lower() == "v
    õru" or murre.lower() == "võro" or murre.lower() == "??võru":
151.                for sona in ainult_murde_sonad:
152.                    teisendused = set(teisendus_seto(sona))
153.                    tolked[sona] = "-"
154.                    for s in teisendused:
155.                        if eeltootlus.kirjakeelsuse_kontroll(s) and len(s) > 1:
156.                            tolked[sona] = s
157.
158.            elif murre.lower() == "saare" or murre.lower() == "saarte":
159.                for sona in ainult_murde_sonad:
160.                    teisendused = set(teisendus_saare(sona))
161.                    tolked[sona] = "-"
162.                    for s in teisendused:
163.                        if eeltootlus.kirjakeelsuse_kontroll(s) and len(s) > 1:
164.                            tolked[sona] = s
165.            else:
166.                sys.stdout.write("\033[1;31m")
167.                print("Ei tunne sellist murret")
168.                sys.stdout.write("\033[0;0m")
169.                return ""
170.
171.            # Väljund sõne loomine. Sisendteksti asendatakse sõnad, mis vajasid tõlkimis
    t
172.            tolke_tekst = ""
```

```
173.           j = 0
174.           for i in range(len(murde_sonad_eemaldatud)):
175.
176.               try:
177.                   if murde_sonad_eemaldatud[i + 1] in ".!?":
178.                       sona_eraldaja = ""
179.                   else:
180.                       sona_eraldaja = " "
181.               except:
182.                   sona_eraldaja = ""
183.
184.               if murde_sonad_eemaldatud[i] == "#":
185.                   if tolked[ainult_murde_sonad[j]] == "-":
186.                       tolke_tekst += "#" + ainult_murde_sonad[j] + "#" + sona_eraldaja
187.                   else:
188.                       tolke_tekst += "$" + tolked[ainult_murde_sonad[j]] + "$" + sona_
    eraldaja
189.                   j += 1
190.               else:
191.                   tolke_tekst += murde_sonad_eemaldatud[i] + sona_eraldaja
192.
193.           return tolke_tekst
194.
195.
196.
197.
198.       sys.stdout.write("\033[1;36m")
199.       print("Programmi kasutamiseks on kolm võimalust.\n"
200.             "1. Käivitada ilma argumentideta. Sel juhul on võimalik otse käsuviipa tõl
    gitavat teksti sisestada\n"
201.             "2. Käsurea argumentidega. Sel juhul on võimalik tekstifaile (.txt ja .doc
    x) sisendiks anda. Argumendid anda kujul \"faili_asukoht murde_nimetus\"\n"
202.             "3. Käsurea argumnedina spetsiaalne muinasjutude kogumik (faili nimi peab
    lõppema \"mjp.docx\". Sel juhul piisab vaid ühest argumnedist, milleks on faili asukoht
    ")
203.       sys.stdout.write("\033[0;0m")
204.
205.       sonastik = set()
206.
207.       tolke_tekst_list=[]
208.       murre = ""
209.
210.       jutude_koguarv = 1
211.       tolgitava_jutuarv = 1
212.
213.       if len(sys.argv)>2:
214.
215.           # Tavaliste tekstifailide tõlkimine
216.           if os.path.isfile(sys.argv[1]):
217.               file=sys.argv[1]
218.               murre = sys.argv[2]
219.               tolke_tekst_list = eeltootlus.get_file_content(file)
220.               tolge = tolgi_tekst(tolke_tekst_list, murre)
221.               print(tolge)
222.           else:
223.               sys.stdout.write("\033[1;31m")
224.               print()
225.               print("Esimene argument pole fail! Kontrollige, sisestatud faili path ol
    eks korrektne")
226.               sys.stdout.write("\033[0;0m")
```

```
227.
228.       elif len(sys.argv)==2:
229.           # Kontoll kas üritatakse tõlikda Kirjandusmuuseumi testdokumente
230.           if "mjp.docx" in sys.argv[1] and os.path.isfile(sys.argv[1]):
231.
232.               # Eraldi funktsioon, mis oskab vastavalt vormistatud .docx faile töödeld
    a
233.               kandid, jutud = docx_test.tagasta_sobiv_vorming(sys.argv[1])
234.
235.               jutude_koguarv = len(jutud)
236.               tolgitava_jutuarv = 0
237.               for i in range(len(kandid)):
238.                   tolgitava_jutuarv = i + 1
239.                   murre = kandid[i]
240.                   jutt = ""
241.                   for ptk in jutud[i]:
242.                       jutt += ptk
243.                   tolke_tekst_list = eeltootlus.sonasta(jutt)
244.
245.                   print(tolgi_tekst(tolke_tekst_list, murre))
246.                   print("Päritolu: ", murre)
247.           else:
248.               print()
249.               sys.stdout.write("\033[1;31m")
250.               print("Sisestatud ainult üks argument!\n"
251.                     "Argumendid anda kujul \"faili_asukoht murde_nimetus\"")
252.
253.       else:
254.           print("Sisesta tõligtav tekst: ")
255.           input_text = input("")
256.           print("Sisesta teksti murre (Saare või Võru/Setu")
257.           murre = input("")
258.           tolke_tekst_list=eeltootlus.sonasta(input_text)
259.           tolge = tolgi_tekst(tolke_tekst_list, murre)
260.           print(tolge)
```

Lisa 4. statistiline_meetod_ibm1.py lähtekood

```
1.  import clize
2.  import json
3.  import docx
4.  from estnltk import Text
5.  import docx_test
6.  import sys
7.
8.  # Edasiarendus repositooriumist: https://github.com/shawa/IBM-Model-1
9.
10. # Statistiline meetod on oma struktuurilt teistest veidi erinev,
11. # kuna see kasutab clize teeki, et käsureaargumente sisse lugeda
12.
13.
14. def get_docx_content(file):
15.     document = docx.Document(file)
16.     list = []
17.     for para in document.paragraphs:
18.         list += lausesta(para.text)
19.     return list
20.
21. def get_txt_content(file):
22.     file = open(file, "r", encoding="utf-8")
23.     list = []
24.     for line in file.readlines():
25.         list += lausesta(line)
26.     return list
27.
28. def sonasta(tekst):
29.     raw_text = Text(tekst)
30.     estnltk_list = raw_text.word_texts
31.     return estnltk_list
32.
33. def lausesta(tekst):
34.     raw_text = Text(tekst)
35.     estnltk_list = raw_text.sentence_texts
36.     return estnltk_list
37.
38. def get_file_content(file):
39.     filetype = file[file.index("."):]
40.     if filetype == ".docx":
41.         content = get_docx_content(file)
42.     elif filetype == ".txt":
43.         content = get_txt_content(file)
44.     else:
45.         print("See failitüüp pole toetatud!")
46.         content = []
47.     return content
48.
49. def tokenize(sentence):
50.     return sonasta(sentence)
51.
52. def translate(tokens, model):
53.     return [model[word] if word in model else word for word in tokens]
54.
55. def tolgi_tekst(tekst, model,jutuarv,koguarv):
56.     print()
57.     print("Alustan teksti tõlkimist!")
```

```
    print(str(jutuarv) + "/" + str(koguarv))
    print()
    dif=0
    all=0
    tolke_tekst = ""
    alg_tekst = ""
    for lause in tekst:
        tokens = tokenize(lause)
        tolge = translate(tokens, model)
        tolke_tekst += tolge[0].capitalize() + " " + " ".join(tolge[1:])
        alg_tekst += tokens[0].capitalize() + " " + " ".join(tokens[1:])
        for i in range(len(tokens)):
            all = all + 1
            if tokens[i].strip()!=tolge[i].strip():
                dif=dif+1
    return tolke_tekst

def main(model_file, sentence):
    sys.stdout.write("\033[1;36m")
    print("Võru keele statistiline masintõlge\n Programmi kasutamiseks on kolm võimalust.\n"
          "1. Käivitada mudelifaili ja tõlgitava lausega, kujul \"mudelifaili_asukoht \"tõlgitav_lause\" (jutumärkides)\" \n"
          "2. Käivitada mudelifaili ja sisendfailiga. Sel juhul on võimalik tekstifaile (.txt ja .docx) sisendiks anda. Argumendid anda kujul \"mudelfaili_asukoht sisendfaili_asukoht\"\n"
          "3. Käivitada mudelifaili ja spetsiaalse muinasjutude kogumikuga (faili nimi peab lõppema \"mjp.docx\". Argumendid anda kujul \"mudelfaili_asukoht sisendfaili_asukoht\"\n")
    sys.stdout.write("\033[0;0m")

    try:
        with open(model_file, 'r', encoding="utf-8") as f:
            model = json.load(f)
    except:
        print("Esimene argument peab olema treenitud mudeli tekstifail")
        return

    try:
        if "mjp.docx" in sentence:
            kandid, jutud = docx_test.tagasta_sobiv_vorming(sentence)
            jutude_koguarv = len(jutud)
            for i in range(len(kandid)):
                tolgitava_jutuarv = i + 1
                murre = kandid[i]
                jutt = ""
                for ptk in jutud[i]:
                    jutt += ptk
                    tolke_tekst_list = lausesta(jutt)
                    if "setu" in murre.lower() or "võru" in murre.lower() or "seto" in murre.lower() or "võro" in murre.lower():
                        print(tolgi_tekst(tolke_tekst_list, model,tolgitava_jutuarv, jutude_koguarv))
                        print("Päritolu: ", murre)
                    return
            else:
                sisu=get_file_content(sentence)
                file_given = True
        except:
            file_given = False

```

```
111.           if not file_given:
112.               if(sentence==""):
113.                   print("Sisesta tõlgitava faili asukoht või tekst mida soovid Võru mu
    rdest tõlkida")
114.                   sentence=input("")
115.                   main(model_file,sentence)
116.                   return
117.               tokens = tokenize(sentence)
118.               translated_tokens = translate(tokens, model)
119.               print("Tõlge: "+" ".join(translated_tokens))
120.               return
121.           else:
122.               print(tolgi_tekst(sisu,model,1,1))
123.
124.
125.       if __name__ == '__main__':
126.           clize.run(main)
```

Lisa 5. Muinasjutt võru murdes „Kuningatütre seitse paari pastlit“

Panti soldat vahti. Soldat vahtis esimese üü. Vahtis teise üü. Vahtis kolmanda üü. Kolmandal üül tuli mure surma ette ja läks kõrtsi. Kõnnib kõrtsi müüdä. Võtab pitsi viina. Vana sandike kõrtsinukan oma kerjakotiga. “Pojake, mis sul viga on?” küsib sant. “Jah, tänä ma elän viil, aga ommen olen ma surnd.” “Räägi mulle.” “Olen kuningatütre seitsme paari pastalde vahis. Olen kaits üüd ärä ollu, iki om uni pääle tükkin ja ei ole nännu midagi.” Sant võtt kotist leeväkoorukese ja sis ütel sedasi: “Noh, kui sul uni pääle tuleb, sis rõbista seda leevakoorukest. Soldat lännu kodu ja lännu kuningatütre usse manu nurka maha. Kuningatüdär tuleb tüdrikuga. Võtab laasi viina unerohuga segatu. Soldat võtab viina ära ja paneb leevakoorukese suhu. Närib jälle leevakoorukest kui rõginal. Kuningatüdär tuleb, kuuleb, et sii makab, ku larab. Võtab tüdriku ja lääb välla seidse paari pastlit ka ühes. Soldatil ää miil: “Nüüd ma nää, kus nad läävad. Nüüd ma lää järgi. Nääb, tüdrukud läävad aida müüdä, uulitsat müüdä ühe õunapuu alla. Võtavad sääl õunu maast ja viskavad üles ja pistavad taskus kah. Ja sis – kadunu. Ei kuskil ega kuskil. Poisil kange irm, et no nüüd ei ole neid kuskil. Odot! Võtt ka õuna maast ja pistab ka taskusse kaits tükki. Kaits tükki taskus ja ühe viskab üles. Sis temä kah maa all. Sääl nääb ku tüdrikud läävad kaugel uulitsat müüdä. Vahepääl istuvad ja panevad jäl´ vahtsit pastlit jalga. Soldat akkab järgi minema. Soldati tii pääl ehen kaklevad põrgulised. Näävad seda soldatit röögivad, et: “roonumiis, roonumiis, lahuta mede tüli!” Soldat viskas nüüd õuna ja puha joosnu kõik õuna manu.

Lisa 6. Saare murdes muinasjutt [20]

Ühekorra olnd üks naine. Ta mees olnd surnd. Naisel olnd üksi sant ning igava elada ning ta tahtand, et mees teda ka eese juure viiks. Ning mees tulnd ka ühe öhta kahe musta täkuga järele. Naine läind suure röömuga mehele seltsi. Saand tee peale söitma, siis mees küsind: „Kas on irm ka, kui kooljas söidab?“. Naine polla sellest midagid aru saand. Niid naine näind (kuupaistega ilm olnd), et varju pole olnd äi ees äga taga. Ning obused söitvad, aga kuulda pole midagi. Ning naisel akkand nii suur irm, et see pole ikka änam öige mees.

Tee ääres olnd üks körts ning naine läind sönna sisse ning ütlend mehele, et ole sa nii kaua vällas. Naine läind sisse ning rääkind körtsimehele äe, et asi on söuke ning söuke, et vanatont on ise kahe musta obusega öues ning oodab mind, et mind pörgu viia. Ladund siis körtsimehega keiksugu kola ukse ede ning lugend mütu issameied peele. Vanatondil läind oodates ka kahtlaseks ning akkand kolistama. Küll tahtand uksest, küll aknast sisse tulla, aga igal pool olnd vägevad ristid ees. Müristand ulk aega, ning viimaks olnd üsna aknast sisse tulemas. Aga siis kukk lauland ning vanakuri kadund kohe äe.

Naine läind omiku koju ning pole änam julgen mötelda, et mees taale järge tuleks. Oleks aga must mees teda äe viind, see oleks teda kindlasti pörgu viind ning siis oleks vanakuradil jälle üks ing rohkem.

Lisa 7. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Silver Vapper**,

(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Murdekeele kirjakeelestamis metoodikad,

(lõputöö pealkiri)

mille juhendaja on **Heiki-Jaan Kaalep**,

(juhendaja nimi)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **11.05.2017**