TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKA TEADUSKOND
Arvutiteaduse instituut

Karl-Oskar Masing

Teadusarvutuse tööaja ennustaja

Bakalaureusetöö (6 EAP)

Juhendaja: Meelis Kull, PhD
Juhendaja: Sven Laur, PhD

Autor: ..... "....." 2013

Juhendaja: ..... "....." 2013
Juhendaja: ..... "....." 2013

Lubatud kaitsmisele
Professor: ..... "....." 2013

Tartu 2013

Sisukord

1 Sissejuhatus

Paljud akadeemilised tööd baseeruvad informatsioonil, kuid seda on sageli tarvis enne järelduste tegemist kas koguda või töödelda. Võttes arvesse tänapäevaseid mõõtmismeetodeid ja andmemahte, on mõistetav, et suurel hulgal juhtudest tuleb mingil hetkel mängu arvuti, mis vastutab kas otseselt mõõtmise eest või mida on tarvis mingil etapil andmete töötlemiseks. Andmaks aluse üldistamiseks, on andmeid vaja palju, mistõttu peab mõõtmiskatseid ja analüüse jooksutama korduvalt. See tõstatab aga probleemi.

Aeg on piiratud ressurss ning seda tuleks kasutada optimaalselt. Kas me jõuame teha soovitud arvutusi/mõõtmisi ettenähtud aja jooksul? Kas me jõuame tegelda mõne muu olulise toiminguga, samal ajal kui arvuti teeb tööd? Teadusarvutuste – ja üldisemalt arvutiprogrammide – korral pole teada nende jooksmise kestus[1]. Teoorias võib pöörduda programmide ajalise keerukuse poole, kuid reaalset vastust see meile anda ei saa, kuivõrd konstantseteks peetavate operatsioonide kestus ja kasutatud algoritmid võivad olla teadmata ning keerulisemate algoritmide analüüs võibki leiduda vaid asümptootilisel juhul – kasutaja aga eeldatavasti lõpmatustega ei tegele.

Käesolev töö pakub süsteemse ning võrdlemisi lihtsa lahenduse, kuidas ennustada suvalise programmijupi tööaega, võttes selleks arvesse varasemate programmijooksutamiste tulemused. Eesmärgi saavutamiseks rakendatakse masinõppe meetodeid, statistikat, kombinatoorikat ja eriilmelisi algoritme ühes arimteetilise avaldise puu andmestruktuuriga.

Töö võib jaotada kolmeks: metoodikaks, metoodika rakendamiseks ning tehtu analüüsimiseks. Metoodikas käsitleme, kuidas annaks probleemi lahendada, rakendamise katab suuresti tööga kaasas olev programm ning teose lõpus arutleme, kui hästi lahendus töötas ning mida võinuks paremini teha.

Enne metoodika juurde asumist katame pinnapealselt teemad, mis leiavad pakutud lahenduses kasutust, ent on tõenäoliselt bakalaureusetaseme üliõpilasele rohkemal või vähemal määral uued. Teemasid on proovitud tutvustada veidi üldisemalt, andmaks esmakordselt tutvujale aimdust, mis võimalusi on vastavas valdkonnas veel, lisaks konkreetselt töös kasutust leidnud meetoditele.

Metoodikale tuginev ennustamistarkvara võiks olla suuteline ennustama, tehes seda mõnesekundilise ajakuluga ning sellise täpsusega, et ennustusest

[1]Käsitlemise all ei ole operatsioonisüsteemide ja veebibrauseri allalaadimiste ennustused, mis arvutavad kuluvat aega hetkeseisundi põhjal ning seega võivad ennustustulemusi muuta.

lähtuvalt oleks kasutajal võimalik edasist tööd planeerida. Lisaks ennustusele võib kasutajat huvitada ka, kuidas tingimuste muutumine mõjutab programmi tööaega, mistõttu oleks hea, kui kasutajal oleks soovi korral võimalik teada saada ka hoomatav ennustamisel kasutatav valem. Kuivõrd ennustus programmi tööaja kohta pole tõenäoliselt tavapärase programmi eesmärk, vaid pigem lisandväärtus, peaks valmiv ennustaja olema lihtsasti külgepoogitav olemasolevatele programmidele.

Ehkki programmide tööaja ennustamise kohta leidub uurimustöid [22], ei ole programme, mis otseselt ennustaksid tööaega, saadaval. Töö lõpufaasis sai aga selgeks, et leidub rakendus [4], mis lahendab andmetöötluse probleemina püstitatud ülesannet sarnaselt töös pakutud lähenemisele. Töö käigus valminud programm säilitab aga unikaalsuse, kuivõrd võimaldab automatiseeritud ennustamist.

Töö on kirjutatud LaTeX-is [5], kasutades selleks Gummi [6] graafilist tekstiredaktorit.

2 Masinõppe tutvustus

2.1 Olemus

Lahendamaks arvuti abil mõnda probleemi, on tavaliseks lähenemiseks mõne algoritmi kasutamine. Algoritmil on sisend ning loogika, mis sisendil fikseeritud protseduure kindlaksmääratud järjekorras ja arv kordi läbi viies tagastab väljundi. Kui algoritm on probleemi suhtes sobivalt valitud, saame ootuspärase tulemuse, mis on lihtsamatel juhtudel ka käsitsi leitav. Halvematel juhtudel on kas algoritm vääralt implementeeritud või probleemi jaoks sobimatu, tuues endaga kaasa ebaloogilise või lausa vale vastuse. Kui meil on aga ettekujutus olemas, kuidas probleemile süstemaatiliselt läheneda, või suudame leida usaldusväärsemast allikast sobilikuma algoritmi, saab vea suurema või vähema vaevaga kõrvaldada, tagades ootuspäraselt töötava probleemilahendaja arvutis.

Mis saab aga siis, kui me ei tea protseduure – või kui neid fikseeritult ei leidugi – mis viiksid sisendist mingisugusesse oodatud väljundisse?

Naiivne oleks väita, et sellisel juhul valime vastuse näiteks juhuslikult muutumispriikonnast. Eriti elavalt kerkib see esile, kui vaadelda mõnda spetsiifilist juhtu – näiteks rämpsposti filtreerimist.

Filtreerides on tarvis tuvastada, kas tegu on rämpspostiga või mitte, seega sisendiks on mingisugusel kujul kiri ning väljundiks piltlikult "ei/jah". Kahjuks pole rämpspostil aga vastet IPv4 *evil bit*'ile [15], mis laseks kergelt formuleerida algoritmi: kirja rämpspostiks olemine sõltub nii situatsioonist kui ka inimesest. Kasutades eelmainitud lähenemist, võiks me juhuslikult märkida saabuva e-kirja kas rämpspostiks või mitte. See poleks aga ilmselgelt aktsepteeritav lahendus, kuivõrd paljud olulised kirjad võiksid kaduma minna.

Sarnastel puhkudel tuleb appi masinõpe. Konkreetse algoritmi asemel saadab distsipliini tabav lause: "*What we lack in knowledge, we make up for in data.*"(tõlkes "Mis meil jääb puudu teadmistes, korvame andmete abil.") [14]. Ehk siis, arvutamiseks kasutatakse olemasolevates andmetes eksisteerivaid mustreid. Eelnevast järeldub ka, et masinõppe edukus sõltub andmete kvaliteedist: kui eelnevad andmed on ebatäpsed või puudub korrapära, pole ka häid tulemusi oodata.

Vaadeldes eelnevat rämpsposti probleemi, on tarvis väikest abi kasutajalt, kes ütleks, millised eksisteerivatest kirjadest on rämpspost ja millised mitte. Selle abil suudab korrektselt valitud ja rakendatud masinõppe mee-

tod luua mudeli, mis võimaldab tulevikus uute kirjade korral mingisuguse täpsusega ennustada, kas tegu on rämpspostiga või mitte. Antud töö raames võib mudelist võib mõelda kui mingist eeskirjast

$$M : (X_1, X_2, .., X_n) \mapsto Y,$$

kus n on lähteandme tunnuste arv, $X_i, i \in \{1, 2, .., n\}$, mingi i-nda tunnuse väärtuste ruum ning Y õppimistulemuste ruum. See aga ei tähenda, et mudelid esituksid alati lihtsa funktsioonina: need võivad olla ka näiteks statistilised jaotused, suured otsustuspuud [3] ja närvivõrgud [8], mis küll antud töö raames käsitlemist ei leia.

On ilmne, et kui kasutaja pole suutnud korrektselt eristada kahte sorti kirju ning on seeläbi andnud "vale teabe" programmile, ei suuda ka programm tulevikus nõnda täpselt ennustada, millise kirjaga on tegu. Ebakvaliteetsete andmete mõju masinõppe meetoditele on uuritud mitmetes artiklites [21, 16].

Alati ei pruugigi meid aga huvitada vaid see, millise tulemuse saame, kui rakendame mudelit mingil andmel. Olles andmete abil mudeli leidnud, võib sama huvitav olla mudelist endast välja loetav informatsioon [14].

Võtame vaatluse alla järgneva fiktiivse andmestiku inimeste kohta, kes on/ei ole abielus.

nimi	...	profiilipilt Facebookis	...	abielus
Jüri	...	jah	...	jah
Margit	...	ei	...	ei
⋮	⋮	⋮	⋮	⋮
Jaan	...	ei	...	ei
⋮	⋮	⋮	⋮	⋮
Liina	...	jah	...	jah

Tabel 1: Fiktiivne andmestik abielu staatusest

Eeldusel, et kirjeldatud andmestikus on tugev seos "profiilipilt Facebookis"="jah" ja "abielus"="jah" ning "profiilipilt Facebookis"="ei" ja "abielus"="ei" vahel, võib masinõppe meetod (kui teised tunnused on kaootilised või pole abieluga väga tugevas seoses) formuleerida mudeli

$$\text{profiilipilt Facebookis} \rightarrow \text{abielus}.$$

Usaldusväärsete andmete korral saab mudelist välja selgitada mõõdetud protsesside olemuse või seose – antud näite puhul tasub kõigil abielluda soovijatel panna Facebooki ilus profiilipilt.

Andmetest rääkides väärib ka nentimist, et ei piisa paarist varasemast mõõtmistulemusest, kuivõrd ei saa olla kindel, et tegu pole eranditega. Mahtude korral on mindud isegi nii kaugele, et on väidetud "*... invariably, simple models and a lot of data trump more elaborate models based on less data.*" [17].

Masinõppe meetodid jagunevad laias laastus kolmeks – juhendajaga, juhendajata ning poolenisti juhendajaga õppimine – sõltuvalt olemasolevatest andmetest.

2.2 Liigid

2.2.1 Juhendajaga õppimine

Juhendajaga õppimise korral võib lähteandmeid kujutada kui vektoreid $(\mathbf{x}, y)$, kus $\mathbf{x}$ kujutab mingite tunnuste hulka ning y mingit $\mathbf{x}$ elementidele vastavusse seatud eritunnust, mille määramist me soovime õpetada. Seega juhendajaga õppimise korral on iga andme puhul teada, milline on "õige" y väärtus vastavale $\mathbf{x}$-le.

y võimalike väärtuste järgi annab juhendatud õppimist jagada omakorda klassifitseerimiseks ja regressiooniks. Klassifitseerimise korral on y väärtused mingist etteantud hulgast. Näiteks eelneva rämpsposti korral on tegu klassifitseerimisprobleemiga, kus võimalikud väärtused on vaid hulgast $\{jah, ei\}$.

Regressiooni korral pole võimalike y väärtuste arv piiratud. Nii on võimalik regressiooni abil hinnata pidevaid suurusi, näiteks aega või raha.

Alternatiivina võib klassifitseerimist ja regressiooni eristada nende mudelite eesmärgi järgi. Klassifitseerimise mudel määrab piltlikult tulemuse olenevalt sellest, millisesse mudeli poolt defineeritud piirkonda tunnuste hulk jääb. Regressiooni korral määratakse tulemus vastavalt sellele, kus tunnuste hulk mudeli määratud joonel/joontel asetseb.

Juhendatud õppimisel on lisaks veel mõned omapärased iseloomujooned. Kuivõrd on teada oodatud y väärtused $\mathbf{x}$ väärtuste korral ning andmestikus võivad esineda mitmed seosed $\mathbf{x}$ ja y vahel, siis annab luua mitmeid mudeleid, mille ennustamise täpsust saab võrrelda statistiliste meetoditega. Sageli kasutatakse hindamiseks valideerimist, millest on juttu peatükis 4.

Juhendatud õppimisel on kaks iseloomulikku probleemi: mudeli ülesobitus ja alasobitus, mis kujutavad endast mudeli üldistamisvõime viga. Ülesobituse

korral treenitakse olemasolevatest (treening-) andmetest liiga detailne mudel, üritades mudeliga määrata võimalikult hästi teadaolevaid $(\mathbf{x}, y)$ paare. Selline lähenemine on eriti ohtlik, kui andmestikus esineb "müra" – andmeid, mis on mõõtmiskatse iseärasuse tõttu eelnevaid andmeid arvesse võttes ootamatud – mis tegelikku seaduspära ei kajasta, ent mõjutab sellest hoolimata mudelit. Programmi tööaja mõõtmisel võib selliseks iseärasuseks olla näiteks protsessori ebatavaliselt suur või väike koormus, mis põhjustab liiga pika või lühikese tööaja.

Alasobituse korral on tegemist mudeli liiga vähese keerukusega. Sellisel juhul ei ole mudel suuteline kirjeldama olemasolevat andmestikku ning seeläbi eksib ka ennustamisel.

2.2.2 Juhendamata õppimine

Juhendamata õppimise korral on lähteandmeteks vaid tunnused – vekotritena kujutades seega $(\mathbf{x})$ – ning pole teada "tegelik" y. Seega ei saa me rääkida, et masinõppe meetod peaks $\mathbf{x}$ korral õigesti ennustama y: tal pole mingit teadmist õigest y-st. Sellise eelinformatsiooniga töötavad masinõppe meetodid tegelevad üldjuhul klasterdamisega, üritades grupeerida mingite tunnuste poolest üksteisega sarnaseid andmeid. Klasterdamine aitab:

a) saada selgust andmete osas,

b) kasutada ära omadust, et paljud andmed on sarnased [24].

Esimesel juhul on võimalik pärast gruppide leidmist andmestikust teha leitu baasil järeldusi või toetada varem eksisteerinud uskumusi/leide. Sellist lähenemist on kasutatud näiteks sotsioloogias, analüüsides erinevate sugude käitumist sõprussuhete moodustamisel [18].

Teisel juhul saab vähendada vaadeldavate andmete arvu, kui neid on näiteks mingi teise masinõppe algoritmi tööks liiga palju (kõigi andmetega võib analüüsiks kuluda liiga palju aega), leides igale moodustunud klastrile "prototüübi", mis iseloomustaks võimalikult hästi kõiki klastri elemente. Kui kõik tunnused kuuluvad Eukleidilisse ruumi, saab prototüübiks võtta näiteks klastri elementide tunnuste keskmiste väärtuste poolt määratud punkti. Nõnda ei pea arvutama enam kõikidel andmetel, vaid piisab klastrite prototüüpidest. Täheldatud on ka, et prototüüpide kasutamine võimaldab andmeid kaoga tihendada, luues tabeli prototüüpidest ning asendades andmed nende klastri prototüübi indeksiga tabelis, ja optimeerida lähimate naabrite leidmist [7],

arvutades välja kaugused vaid naabriteni, mille klastrite prototüübid on piisavalt lähedal [24].

Lisaks masinõppele leiavad juhendamata õppimise meetodid rohket kasutust ka andmekaeves.

2.2.3 Poolenisti juhendajaga õppimine

Poolenisti juhendajaga õppimise korral on mingi osa andmetest kujul ($\mathbf{x}$,y) ning ülejäänud osa kujul ($\mathbf{x}$), ehk vaid kindlal hulgal andmetel on teada, milline on tunnuste $\mathbf{x}$ väärtuste korral tunnus y. Selliste andmestike põhjuseks võib olla näiteks tunnuse y määramise keerukus/hind või andmete rohkus, mille korral võib kõikide y väärtuste leidmine/talletamine olla ebaoptimaalne. Omades andmestikku, kus vaid andmetel A pole leitud y, saame A abil leida juhendamata õppimise meetodite – näiteks klasterdamise – abil probleemi piiritlevat lisainformatsiooni, mida annab kasutada ülejäänud andmetel juhendatud õppimise meetodite tulemuse parandamiseks.

2.3 Kasutusvaldkond

Varasemalt on mainitud rämpsposti filtreerimist, kuid see pole ainus, kus masinõpe kasutust leiab. Masinõpet on lisaks edukalt rakendatud hääletuvastuses, raalnägemises, robotite juhtimises [19], pankades pettuste tuvastamiseks, tööstuses optimeerimiseks ja juhtimiseks, meditsiinis ekspertsüsteemides diagnooside määramisel [14] ning ohtratel teistel aladel. Nüüdsest loodetvasti rohkem ka programmide tööaja ennustamisel.

3 Regressioon

Regressioon (või täpsemalt regressioonanalüüs) tegeleb muutujate vaheliste seoste hindamisega. See hõlmab paljusid erinevaid meetode, mis võimaldavad leida seose mingi huvipakkuva muutuja (sõltuva muutuja) y ja muutujate $x_1, x_2, \ldots, x_n$ vahel [12]. Seosed leitakse olevasolevate muutujate väärtuste abil. Mõtleme edaspidi muutujatest $y, x_1, x_2, \ldots, x_n$ kui tunnustest ning eeldame käesoleva töö raames, et tunnused omandavad vaid reaalarvulisi väärtusi.

Definitsioon. *Nimetame* ***mudelkujuks*** $G(x_1, x_2, \ldots, x_n)$ *tunnuse* y *ja tunnuste* x_i *vahelist seost kirjeldavat lineaarkombinatsiooni*

$$y = G(x_1, x_2, \ldots, x_n) = \sum_{j=0}^{c} \alpha_j K_j(x_1, x_2, \ldots, x_n),$$

kus funktsiooni $K_j(x_1, x_2, \ldots, x_n)$ *nimetame* ***komponendiks*** *ning mille korral kõik komponendid on erinevad – st, kui* $K_a(x_1, x_2, \ldots, x_n) \equiv K_b(x_1, x_2, \ldots, x_n)$*, siis* $a = b$ *– ning* α_j *on muutuja.*

Kokkulepe. *Olgu* $K_0(x_1, x_2, \ldots, x_n) \equiv 1$*, mis hakkab edaspidi kujutama vabaliiget.*

Definitsioon. *Nimetame* ***mudeliks*** $M(x_1, x_2, \ldots, x_n)$ *mudelkuju, mille kõik muutujad* α_j *on väärtustatud reaalarvudega.*

Seega taandub seose leidmine tunnuse y ja tunnuste x_i vahel mudeli leidmisele. Reaalsete andmete korral on aga vähetõenäoline, et mõistliku keerukusega mudel seaks perfektselt tunnuste x_i väärtustele vastavusse y väärtuse, kuna peaaegu eranditult esineb andmetes müra, mistõttu ei eeldata ideaalset kirjeldamist. Olgu meil tunnuste x_i väärtusi kujutav andmemaatriks $X = (x_{ji}) \in Mat(m, n)$ ning veeruvektor $Y = (y_{j1}) \in Mat(m, 1)$, kus j fikseerib katse ning i tunnuse. Sellisel juhul rahuldutakse k-nda katse korral, kui

$$y_{k1} = M(x_{k1}, x_{k2}, \ldots, x_{kn}) + \varepsilon_k,$$

kus ε_k tähistab mudeli viga k-nda mõõtetulemuse korral.

Definitsioon. *Nimetame vektorit* $(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m)$ *mudeli* ***veavektoriks****.*

Regressioonimeetodid võib jaotada kaheks:

- parameetrilised,
- mitteparameetrilised.

3.1 Parameetrilised regressioonimeetodid

Parameetrilistel regressioonimeetoditel on eeldefineeritud mudelkujud ning regressiooni käigus on tarvis leida vaid sobivaimad muutujate väärtused ehk parameetrid [12]. Lihtsaimateks näideteks on lineaarse regressiooni meetodite mudelkujud, kus andmete põhjal otsitavateks parameetriteks on koefitsiendid α_i, $i \in \mathbb{N}$, mis on üksteisega vaid liitmistehete abil seotud. Järgnevalt tutvustame konkreetsuse huvides kahe lineaarse regressiooni meetodi mudelkuju:

- lihtne lineaarne regressioon – $y = G(x) = \alpha_0 + \alpha_1 x$;
- mitmene lineaarne regressioon – $y = G(x_1, x_2, \ldots, x_n) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$.

Kuivõrd tunnuseid on alati lõplik arv, saab parameetrilise regressiooni mudeleid võrdlemisi mõistliku vaevaga kujutada – eriti, kui mõned koefitsientidest osutuvad nulliks.

On ka ilmne, et parameetriliste regressioonimeetodite kasutamisel peab eeldama, et andmed on vastava mudeli abil kirjeldatavad – vastasel juhul on paratamatu, et mudel eksib palju.

3.2 Mitteparameetrilised regressioonimeetodid

Mitteparameetriliste regressioonimeetodite korral ei ole mudelkujud teada, mistõttu peab need andmetest tuletama [9]. Seetõttu on tegu parameetrilisest regressioonist veidi keerulisema probleemiga, kuna teadmata pole mitte ainult komponentide $K_j(x_1, x_2, \ldots, x_n)$ kordajad, vaid ka komponendid ise. Nimetatud probleemi lahendab näiteks mitteparameetriline multiplikatiivne regressioon [9], mis jaotab andmestiku osadeks ning leiab lähedal paiknevate andmepunktide korral lokaalsed mudelid ja koostab lõpuks nende abil ühtse mudeli.

Sellise lähenemise suurimaks probleemiks on saadava mudeli tõlgendatavus – eriti kui tunnuste väärtused on suures vahemikus – kuivõrd tulemust võib ette kujutada kui tükiti funktsiooni. Probleemi lahendamiseks on välja töötatud erinevaid meetodeid ning mudelkujusid. Üheks selliseks mudelkujuks on aditiivne mudel [1], mis avaldub kujul

$$y = G(x_1, x_2, \ldots, x_n) = \alpha_0 + \sum_{i=1}^{n} f_i(x_i),$$

kus f_i on mingi sile funktsioon ehk diferentseerub vaadeldava piirkonna igas punktis.

4 Mudelite võrdlemine

Mudelite võrdlemine on oluline etapp, eriti kui andmestikku tegelikult kujutavad mudelkujud on teadmata (ehk tegu on mitteparameetrilise regressiooni probleemiga) või ideaalne mudel osutub näiteks keerukuse tõttu soovimatuks. Sellisel juhul on meil mitmed potentsiaalsed mudelid, mis võiksid valituks osutuda, kuid meie soov on leida kandidaatide seast mudel, mis kannataks võimalikult vähe üle -ja alasobituse käes.

Kui andmestikku tegelikult kujutav mudelkuju on teada ning andmetes puudub müra, võib piisata vaid mudeliti veavektorite elementide summeerimisest ja summade ($\sum_{k=1}^{m} |\varepsilon_k|$) või nende baasil leitava "keskmise vea" ($\frac{\sum_{k=1}^{m} |\varepsilon_k|}{m}$) võrdlemisest.

Reaalsuses pole aga sageli tegelik mudelkuju teada või tegelikku mudelkuju ei anna 100%-lise täpsusega leida, kuivõrd teada on vaid mingi osa kõikvõimalikest $(x_1, x_2, \ldots, x_n, y)$ väärtuste kombinatsioonidest ning needki on vähema või rohkema müraga. Seeõttu on välja töötatud meetodid, mis suudavad hindamisel arvesse võtta, et fikseeritud treeningandmestikul treenitud mudel ei ole ilmtingimata suutlik uute andmete $(x_1, x_2, \ldots, x_n)$ korral ootuspäraselt leidma neile vastavat väärtust y. Järgnevalt tutvustame põgusalt kolme meetodit.

Bayesi informatsiooni kriteerium (BIC) on statistiline lähenemine mudelile hinnangu andmiseks, mille korral võetakse arvesse tõepära, et vaadeldavad andmed on vastava mudeli poolt genereeritud, mis vähendab alasobituse ohtu, ning karistatakse parameetrite arvu eest, mis vähendab ülesobituse ohtu. Bayesi informatsiooni kriteerium avaldub peatüki 3 muutujanimede tava kohaselt

$$BIC = -2 \ln L + c \ln(m),$$

kus c on lineaarse regressiooni korral komponentide $K_j(x_1, x_2, \ldots, x_n)$ arv [2], m on mõõtmistulemuste arv ning L on tõepära, et andmed on juhuslikult genereeritud vastava mudeli poolt. Mida väiksem on avaldise väärtus, ehk mida lihtsam ja täpsem on mudel, seda parema mudeliga on tegu.

Minimaalse kirjelduse printsiip (MDL) on informatsiooniteoreetiline lähenemine, kus probleemi vaatadakse andmete saatmise poole pealt. Olgu meil hulk andmeid, mida soovime saata mingil digitaalsel viisil bittidena. Meie eesmärgiks on leida mudel, mille abil andmeid kirjeldades kuluks

bitte minimaalselt. Mudeli abil andmete kirjeldamisel kulub bitte nii mudelile kui ka veavektorile – mida enam mudel mingi mõõtmistulemuse kirjeldamisel eksib, seda rohkem bitte kulub vastava vea kujutamiseks. Selline lähenemine vähendab ülesobituse ohtu, kuivõrd keerulisem mudel võtab rohkem bitte, ning alasobituse ohtu, kuna andmeid halvemini kirjeldava mudeli korral kulub rohkem bitte veavektori kujutamisele. Sobilik mudel oleks seega selline, mis oleks ühtaegu mõõdukalt täpne ja lihtne. Kuivõrd MDL soosib sarnaseid mudeleid nagu BIC ning on võimalik näidata, et andmete kasvades on MDL ja BIC avaldised sarnaselt piiratud, on hoiatatud nende äravahetamise eest, kuivõrd üldiselt annavad nad erinevaid tulemusi [20].

Valideerimise korral jagatakse andmestik kaheks: treening -ja valideerimisandmestikuks. Treeningandmestikul "treenitakse" ehk leitakse treeningandmestikku kirjeldav mudel. Valideerimisandmestikul kontrollitakse, kui hästi treeningandmestikust sõltumatul valideerimisandmestikul mudel töötab, imiteerides tundmatuid andmeid, millega mudel puutub kokku ka ennustades või klassifitseerides. Nõnda vähendadakse ülesobituse ohtu.

5 Ennustamise metoodika

Programmi tööaega ennustades on vaja mingit lähtepunkti, mille abil saaks eristada erinevaid programmi jooksutamisi: pelgalt eelnevate tööaegade teadmisest ei piisa, kuivõrd pole mingit võimalust siduda ennustada soovitavat programmi jooksutamist mingi varemleitud ajaga.

Selle tarbeks on kasutusel programmi jooksutamisel olnud parameetrid. Parameetriteks võivad olla mistahes arvulised väärtused, mis on iseloomulikud just mingil kindlal ajal jooksutatud fikseeritud programmile. Kui tegu on sorteerimisalgoritmi implementatsiooni tööaja ennustamisega, on kõige ilmsemaks oluliseks parameetriks sorteeritavate arv. Rolli võivad mängida ka teised tegurid, nagu näiteks sorteeritavate talletamise täpsus bittides (kui on tegu arvudega) või Boole'i väärtusi omavad "sisend eelnevalt sorteeritud" ja "sisendis esinevad kordused". Kuna protsessi tööaeg varieerub erinevate arvutite korral suuresti, on ilmne, et parameetriteks peaksid olema ka kõikvõimalikud arvutinäitajad. Kuna nende väärtusi või isegi olemasolu ei pruugi lihtkasutaja aga teada, on soovituslik, et andmed oleksid kogutud ühel masinal, mille korral arvutinäitajad kaotavad tähtsuse.

Parameetreid kasutades taandub ennustamine parameetrite ja aja vahel eksisteeriva seose leidmisele. Pidades nii parameetreid kui ka aega tunnusteks, saab kasutada statistikute poolt väljatöötatud meetode. Neist väärib eelkõige mainimist regressioon -ja faktoranalüüs. Faktoranalüüs üritab aga olemasolevate tunnuste vahel leida seoseid tundmatute tunnuste abil, mistõttu ei paku see antud ülesande kontekstis huvi. Kasulikuks osutub aga regressioon.

Peatükis 3 sai tutvustatud erinevat liiki regressioone. Parameetriline regressioon oma eeldusega, et tunnus avaldub teiste kaudu mingi kindla valemikuju abil, ei ole käesoleva probleemi lahendamiseks piisavalt efektiivne, kuna eeldame, et meil pole kuluva aja kohta eelinformatsiooni: seega ei saa me teada, kuidas aeg võiks avalduda.

Mitteparameetrilise regressiooni korral on aga sageli keeruline mudelit hoomata. Ehkki seda omadust on püütud parandada näiteks aditiivse mudeli abil, mida on lühidalt tutvustatud osas 3.2, ei saa selles tunnuste $x_i, x_j, i \neq j$, vahel kirjeldada liitmisest keerulisemaid seoseid. Programmides kasutatakse aga sageli kahekordseid tsükleid, mille tööaeg avaldub korrutisena.

Korvamaks eelmainitud lähenemiste puudujääke, toome aditiivsest mudelist inspireerituna sisse uue mudelkuju

$$G(x_1, x_2, \ldots, x_n) = \alpha_0 f_0(\mathbf{x}) + \alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x}) + \ldots + \alpha_m f_m(\mathbf{x}), \quad (5.1)$$

kus $m+1$ on liidetavate komponentide arv, α_i, $0 \leq i \leq m$, on muutujad, ning f_j, $0 \leq j \leq m$, on funktsioonid, mis võtavad argumentideks $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ ja avalduvad argumentide ning nende logaritmide korrutisena.

Kuna programmides on sageli konstantse ajakuluga osad, mida ei mõjuta parameetrite väärtus, on mõistlik käsitleda edaspidi mudelkuju (5.1) erikuju, mille korral fikseerime funktsiooni f_0 selliselt, et $f_0(\mathbf{x}) \equiv 1$. Seega töötame edaspidi mudelkujuga

$$G(x_1, x_2, \ldots, x_n) = \alpha_0 + \alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x}) + \ldots + \alpha_m f_m(\mathbf{x}). \tag{5.2}$$

Kahjuks puuduvad meil aga tavapärased vahendid, et leida aega kirjeldama parim võimalik (5.2) kujul mudel. Aditiivsete mudelite korral rakendatavad statistilised meetodid võimaldavad iteratiivselt leida erikujul olevaid koefitsientideta funktsioone $f_k(x_k)$, $1 \leq k \leq n$, kuid teadmata, milliseid ja kui palju parameetreid funktsioon $f_k(\mathbf{x})$ võtab ning millise kujuga võiks $f_k(\mathbf{x})$ olla, ei ole neist kasu.

Parima mudeli leidmiseks pakume välja mudelite ruumis genereerimise ja testimise. Esitades ülesande sellisel kujul, peame lahendama 3 suuremat probleemi:

(1) mudelite genereerimine,

(2) mudelite võrdlemine,

(3) genereerimise ja testimise peatumine.

Mudelite genereerimine tagab mudelite kandidaadid, mille hulgast leitakse mudelite võrdlemise etapil parim mudel. Parimat leitud mudelit annab seejärel juba rakendada ennustatava programmi parameetritele tööaja hindamise eesmärgil.

5.1 Genereerimise ja testimise peatumine

Mudelite ruumis on mudeleid on lõpmatult palju. Mõtleme kasvõi funktsioonidest $g_i(x) = x^i$ kui lihtsatest mudelitest, kus $i \in \mathbb{N}$.

Seega on mõeldamatu genereerida nii kaua, kuni kõik mudelid on läbi vaadatud. Lahenduseks võib genereerida seni, kuni on mingi hulk mudeleid läbi vaadatud või kuni on möödunud mingi aeg. Viimane on eelistatud, kui

ennustamist kasutatakse vaid mõõdetava programmi lisana, andmaks mingi ligilähedase hinnangu oodatavale tööajale – see garanteerib, et ennustaja ei tööta kauem, kui mõõdetava programmi kasutajal kannatust jagub.

5.2 Mudelite genereerimine

Mudelite genereerimise eesmärgiks on leida potentsiaalselt aega kujutavad mudelid mudelkujuga (5.2). Geneeritud mudelitel on oluline unikaalsus ja asjakohasus: vaadeldavate mudelite hulk on aja või arvu poolt piiratud, mistõttu ei saa lubada korduseid – mida enam erinevaid mudeleid võrrelda, seda suurem on tõenäosus leida andmete kirjeldamiseks hea mudel. Lisaks tuleks ka genereeritavad mudelid kuidagi prioritiseerida: käesolevas töös panustatakse eelkõige lihtsamatele vähemate komponentidega mudelitele, mis koosnevad liitmisest, korrutamisest ja logaritmimisest. Liitmine iseloomustab hästi programmide lineaarsust: koodi jooksutatakse rida rea haaval ning iga rida lisab mingi aja kogu programmi tööajale. Korrutamise abil avalduvad mitmekordsed tsüklid ning logaritmina sageli näiteks optimaalsed sorteerimised ja puudes liikumine.

Mudeleid mudelkujuga (5.2) genereerides annab ülesande jagada kaheks: esmalt saab leida funktsioonid f_i ning nende ja andmete baasil seejärel koefitsiendid α_j, mis seavad leitud funktsioonide komplekti võimalikult hästi andmeid kirjeldama.

5.2.1 Fuktsioonide komplektide leidmine

Läheneme probleemile mitmeetapiliselt. Tähistame sümboliga x valemis alamavaldist, mis asendatakse mingi parameetriga, sümboliga H_i i-nda etapi tulemust ning lepime kokku, et xx tähistab kahe järjestikuse parameetri korrutamist ja + tavapärast liitmist. Fikseerime **esimeses etapis** mingi arvu $s \in \mathbb{N}$ ning leiame sümbolite $\{\text{“x”}, \text{“+”}\}$ kõikvõimalikud järjestused pikkusega $1 \dots s$, nii et sümbol “+” ei oleks esimesel või viimasel kohal ning et ei esineks kõrvuti kahte sümbolit “+”. Kui $s = 3$, siis

$$H_1 = \{\text{“x”}, \text{“xx”}, \text{“xxx”}, \text{“x+x”}\} \tag{N1}$$

Olles leidnud sõne kujul kõikvõimalikud avaldatavad valemite kujud s piires, peame teises etapis fikseerima, millised x-id milliste parameetritega asendatakse. Olgu n parameetrite arv ning tähistagu x_i i-ndat parameetrit, $1 \leq i \leq$

n. Arvestades, et parameetrid omandavad reaalarvulisi väärtusi ning korrutamine on reaalarvude hulgal kommutatiivne, pole mõtet asendada näiteks sõnet "xxx" eraldi nii sõnega "$x_1x_1x_2$" kui ka "$x_1x_2x_1$". Selleks leiame asendatud parameetrid nii, et kui sõnes esineb "x_ix_j", siis $i \leq j$. Olgu näiteks $n = 2$. Sellisel juhul on teise etapi tulemusena näite (N1) hulk teisendatud kujule

$$H_2 = \{\text{“}x_1\text{”}, \text{“}x_2\text{”}, \text{“}x_1x_1\text{”}, \text{“}x_1x_2\text{”}, \text{“}x_2x_2\text{”}, \text{“}x_1x_1x_1\text{”}, \text{“}x_1x_1x_2\text{”},$$
$$\text{“}x_1x_2x_2\text{”}, \text{“}x_2x_2x_2\text{”}, \text{“}x_1 + x_1\text{”}, \text{“}x_1 + x_2\text{”}, \text{“}x_2 + x_1\text{”}, \text{“}x_2 + x_2\text{”}\} \qquad \text{(N2)}$$

Seega **teise etapi** lõpuks oleme leidnud sõne kujul kuni s sümboliga valemid, mis sisaldavad parameetrite korrutamist ja liitmist ning korrutistes ei järgne suurema indeksiga parameetrile väiksema indeksiga parameeter. Nüüd lisame valemitele naturaallogaritmid.

Kuivõrd reaalarvude korrutamine on kommutatiivne, pole tarvis erinevalt käsitleda juhte, mil mingi parameeter on logaritmitud kas ühel või teisel positsioonil: näiteks $x_1 \ln(x_2)x_2 \equiv x_1x_2 \ln(x_2)$. Seetõttu hoolime vaid, kui mitu korda mingis korrutises mingit parameetrit logaritmitakse. Selleks jätame mugavuse mõttes kõrvale harjumuspärase naturaallogaritmi tähistuse "ln" ning asendame selle sümboliga "$\bullet$"[2], kus mingi parameetrite x_i järjestuse ees olev sümbolite "$\bullet$" arv tähistab, mitut järgnevat parameetrit x_i me logaritmime. Kui soovime näidata, et korrutises $x_1x_1x_1x_2x_3$ logaritmime parameetrit x_1 kaks korda ning x_3 ühe korra, tähistame seda sõnega "$\bullet\bullet x_1x_1x_1x_2\bullet x_3$", mis oleks tavatähistuses identne korrutisega $\ln(x_1)\ln(x_1)x_1x_2\ln(x_3)$.

Kolmanda etapi ülesandeks jääb leida kõik kombinatsioonid, kuidas korrutistes olevaid parameetreid logaritmida ehk leida iga teise etapi tulemuse h korral kõikvõimalikud viisid seada iga k korrutatud parameetri x_i ette $0 \ldots k$ sümbolit "$\bullet$". Jätkates näitega (N2), oleks kolmanda etapi lõpuks

$$H_3 = \{\text{“}x_1\text{”}, \text{“}\bullet x_1\text{”}, \text{“}x_2\text{”}, \text{“}\bullet x_2\text{”}, \text{“}x_1x_1\text{”}, \text{“}\bullet x_1x_1\text{”}, \text{“}\bullet\bullet x_1x_1\text{”},$$
$$\text{“}x_1x_2\text{”}, \text{“}\bullet x_1x_2\text{”}, \text{“}x_1\bullet x_2\text{”}, \text{“}\bullet x_1\bullet x_2\text{”}, \ldots, \text{“}x_1 + x_2\text{”},$$
$$\text{“}\bullet x_1 + x_2\text{”}, \text{“}x_1 + \bullet x_2\text{”}, \text{“}\bullet x_1 + \bullet x_2\text{”}, \ldots\}. \qquad \text{(N3)}$$

Praegusel tulemusel on aga üks tõsine puudus: nimelt, olgu näiteks $s = 3$. Siis saame esimesel etapil võimalikeks kujudeks nii "x" kui ka "$x+x$". Asendades x järgnevatel etappidel näiteks x_1-ga, saame kolmandal etapil teiste

[2]Implementatsioonis on $\bullet$ asemel kasutatud tagurpidi ülakoma, aga kuna see sarnaneb sõnet piiritlevate jutumärkidega, on loetavuse nimel tehtud asendus.

hulgas ka sõne kujul valemid "x_1" ja "$x_1 + x_1$". Kuivõrd $x_1 + x_1 = 2x_1$ ning arvestades, et pärast sõnede funktsioonideks parsimist antakse funktsioonid koefitsientide leidmisel edasi lineaarsele regressioonile, on nimetatud funktsioonide poolt loodavad mudelid sama täpsed, mistõttu pole tarvis mõlemaga testida. Sama täpsus tuleneb sellest, et kui lineaarse regressiooni käigus selgub, et andmeid kirjeldab funktsioon x_1 kõige paremini koefitsiendiga α_0, siis ilmselgelt $2x_1$ peab andmeid kirjeldama kõige paremini koefitsiendiga $\frac{1}{2}\alpha_0$.

Probleemi lahendamiseks on tarvis eemaldada korduvad H_3 elemendid, kus "korduvad" on suvalises järjekorras samu liidetavaid omavad, kusjuures liidetavate kordumine pole oluline. Selleks defineerime sõne kujul olevatel elementidel kanoonilise kuju.

Definitsioon. *Olgu "" $\prec$ "$\prec$" $\prec$ "x_2" $\prec$ ... $\prec$ "x_n" $\prec$ "$\bullet$". Nimetame $\forall e \in H_3$ kanoonilisel kujul olevaks, kui e kujutab ühte korrutist või liidetavaid on ≥ 2 ning need on leksikograafiliselt järjestatud nii, et kõik liidetavad on esindatud ühekordselt.*

Eemaldame **neljandas etapis** kõik H_3 elemendid, mis ei ole kanoonilisel kujul. Näiteks on kanoonilisel kujul "$x_2x_2 + \bullet x_1x_1 + \bullet x_1x_2x_2$", kuid mitte "$\bullet x_1x_1 + \bullet x_1x_2x_2 + x_2x_2$". Paneme tähele, et kuivõrd sõned ei sisalda enam korduvaid liidetavaid, saame ka pärast lineaarset regressiooni unikaalsed valemid, kui hoiame H_4-s kõiki sõne kujul olevaid elemente ühekordselt.

Omades mudelite suhtes eelistusi ning arvestades, et läbi jõuab vaadata vaid teatud osa, on mõistlik **viienda etapina** H_4 elemendid mingi vastava kriteeriumi alusel järjestada. Seame igale H_4 elemendile, mis kujutab valemit, vastavusse keerukuse hinnangu. Selleks arvestame valemis esinevate erinevate parameetrite arvu s_1, parameetrite koguarvu s_2, liitmistehete arvu s_3 ning logaritmimiste arvu s_4.

Näiteks valemi

$$x_1x_1 \ln(x_2)x_3 + x_1x_2 + x_4$$

korral $s_1 = 4$, $s_2 = 7$, $s_3 = 2$ ja $s_4 = 1$. Leitud väärtustest saame hinnangu avaldise

$$\sum_{i=1}^{4} w_i s_i$$

abil, kus w_i on s_i-le omistatud kaal.

Kuivõrd mudelite genereerimisel on tarvis väljastada funktsioonide hulk, piisab **kuuendas etapis** H_5 elementide tükeldamisest liitmistehetelt.

5.2.2 Koefitsientide leidmine

Fikseerides funktsioonid $f_j(\mathbf{x})$, $1 \leq j \leq m$, on tarvis mudelkuju (5.2) jaoks leida koefitsiendid α_k, $0 \leq k \leq m$. Kuivõrd soovime, et saadav mudel kirjeldaks hästi ka andmeid, peavad koefitsiendid olema valitud optimaalselt. Olgu meil andmed D, kus $\forall d = (\mathbf{x}, y) \in D$ ning y on x_i-de kaudu avaldatav tunnus (antud töö kontekstis aeg), $1 \leq i \leq n$.

Defineerime kujutise

$$F\colon\ (\mathbf{x}, y) \mapsto (1, f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}), y)$$

ning uue andmestiku

$$D^* = \{F(d) \mid d \in D\}\,.$$

Esialgse probleemi suhtes on D^* avaldatav mitmese lineaarse regressiooni mudelina

$$G(1, u_1, u_2, \ldots, u_m) = \alpha_0 + \alpha_1 u_1 + a_2 u_2 + \ldots + \alpha_m u_m,$$

kus $u_j = f_j(\mathbf{x})$. Paneme tähele, et otsitavateks on veel vaid koefitsiendid α_k, olles seega probleemi taandanud mitmesele lineaarsele regressioonile, mille korral on teada, kuidas koefitsiendid leida.

5.3 Mudelite võrdlemine

Pärast mudeli M leidmist soovime anda sellele hinnangu, oskamaks seda teiste leitud mudelitega võrrelda. Mudelite võrdlemise aluseks on valideerimine, mis lubab hinnata mudeli üldistusvõimet. Selle tõttu jagame enne treenimist andmestiku D kaheks: testandmestikuks D_{test} ja treeningandmestikuks $D_{treening}$. Viimane osaleb ka mudeli genereerimisel.

Näidaku

$$E(\mathbf{x}, y) = \frac{y}{M(\mathbf{x})}$$

ennustuse ja tegeliku väärtuse suhet. Siis saame suhete jada

$$E_i = \frac{y_i}{M(\mathbf{x}_i)}, \qquad i = 1, \ldots, k.$$

Ideaalse mudeli korral oleksid kõik suhted lähedal ühele. Kuna tavaliselt üritavad regressioonimeetodid leida lahendi, mille korral suhete keskmine on

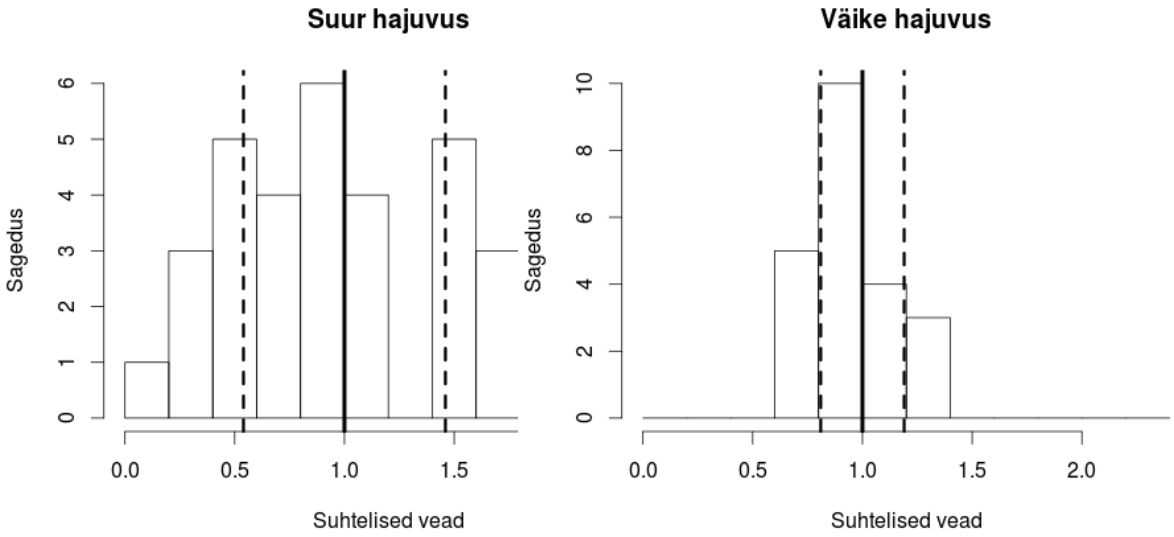
ligikaudu 1, siis näitab sobituse kvaliteeti vigade hajuvus, mida saab kujutada variatsiooni abil.

$$\mathbf{Var}(E_i) = \frac{1}{k}\sum_{i=1}^{k}(E_i - \bar{E})^2,$$

kus $\bar{E}$ on suhete keskmine väärtus

$$\bar{E} = \frac{1}{k}\sum_{i=1}^{k} E_i$$

Illustreerime mõtet graafiliselt.



(a) Suur hajuvus (b) Väike hajuvus

Joonis 1: Suhteliste vigade hajuvus,

Nagu jooniselt 1.a näha, on suurema hajuvuse korral ka mudeli ennustused tegelikest väärtustest kaugel.

Enamasti vaadatakse mudelite võrdlemisel ruutkeskmist viga

$$MSE = \frac{1}{k}\sum_{i=1}^{k}(y_i - M(\mathbf{x}_i))^2,$$

kuid kuivõrd meil on kasutusel suhteline viga, ei osutu see kõige paremaks. Mõtleme näiteks kahest punktist (x_1, y_1) ja (x_2, y_2), kus $|y_1 - M(x_1)| = |y_2 - M(x_2)|$. Kui ühe väärtused on teise väärtustest väiksemad, on jagatised tunduvalt erinevad.

Vaadata võib ka suhtelise vea hajuvust

$$\delta_i = \frac{y_i - M(x_i)}{y_i},$$

mis annab mõneti teistsuguse tulemuse kui suhete hajuvust mõõtev variatsioon.

Kuna suhteline viga ei saa omandada negatiivseid väärtusi (seame mistahes mudelile nõude, et ei ennustataks negatiivseid tööaegu) ning jääb sageli võrdlemisi väikesesse vahemikku, kasutame standardhälbe hinnangu leidmisel log-normaaljaotust, mis negatiivseid väärtusi ei oma. See aga ei tähenda, et me eeldaks, justkui $E_1, \ldots, E_k$ olekski nimetatud jaotusega. Me kasutame seda vaid standardhälbe hindamisel. Selline näiliselt statistika häid tavasid ignoreeriv lähenemine on masinõppes ka üldisemalt kasutusel [23].

Kuivõrd katsetulemuste Z log-normaaljaotuse keskväärtuse μ ja standardhälbe σ hinnangud avalduvad kujul

$$\mu = \frac{\sum_{z \in Z} \ln z}{|Z|},$$

$$\sigma = \sqrt{\frac{\sum_{z \in Z} (\ln z - \mu)^2}{|Z|}},$$

siis tehes vastavad asendused, saame

$$\mu_{treening,M} = \frac{\sum_{((\mathbf{x},y) \in D_{treening}} \ln E(\mathbf{x}, y)}{k}$$

ja

$$\sigma_{treening,M} = \sqrt{\frac{\sum_{(\mathbf{x},y) \in D'_{treening,M}} (\ln E(x, y) - \mu_{treening,M})^2}{|D_{treening}|}}.$$

Me ei saa aga piirduda pelgalt treeningandmetel suhteliste vigade standardhälvete hindamisega, kuivõrd sellisel juhul osutuvad sageli valituks ülesobitunud mudelid. Selle vältimiseks on kasutusele võetud meetodid, mis karistavad liigse

keerukuse eest – näiteks minimaalse kirjelduse printsiip ja Bayesi informatsiooni kriteerium – eelistades mõõdukalt täpset ja lihtsat mudelit. Sellisel juhul võib aga aset leida hoopis alasobitus, kuivõrd võib juhtuda, et keerulisi mudeleid karistatakse nende keerukuse pärast, ehkki on ära õppinud tegeliku parameetrite seose ning suudavad täpselt ennustada ka tundmatute andmete korral.

Lahenduseks pakume välja leida treeningandmetelt $D_{treening}$ ekvivalentsiklassiti parimad mudelid.

Olgu $\sim$ ekvivalentsiseos kõikide vaadeldavate mudelite hulgal U_M, mille korral $A \sim B$, $A, B \in U_M$ siis, kui mudelitel A ja B on sama palju liidetavaid ning liidetavates on kokku sama palju parameetreid.

Valime igast ekvivalentsiklassist välja mudeli M, mille puhul on standardhälbe hinnang log-normaaljaotuse korral $D_{treening}$ andmetel vähim. Peaks ühes ekvivalentsiklassis leiduma mitu minimaalse standardhälbe hinnanguga mudelit, valime juhuslikult ühe.

Omades ekvivalentsiklasside parimate mudelite hulka pärast genereerimise ja testimise peatumist, kasutame taas esialgset lähenemist võrdlemisele, kuid seekord testandmestiku D_{test} ja parimate leitud mudelite korral. Leiame mudelite suhteliste vigade keskväärtuse

$$\mu_{test,M} = \frac{\sum_{(\mathbf{x},y)\in D_{test}} \ln E(\mathbf{x}, y)}{|D_{test}|}$$

ja standardhälbe hinnangu

$$\sigma_{test,M} = \sqrt{\frac{\sum_{(\mathbf{x},y)\in D_{test}} (\ln E(\mathbf{x}, y) - \mu_{test,M})^2}{|D_{test}|}}$$

log-normaaljaotuse korral. Valituks osutub mudel, millel on $\sigma_{test,M}$ minimaalne.

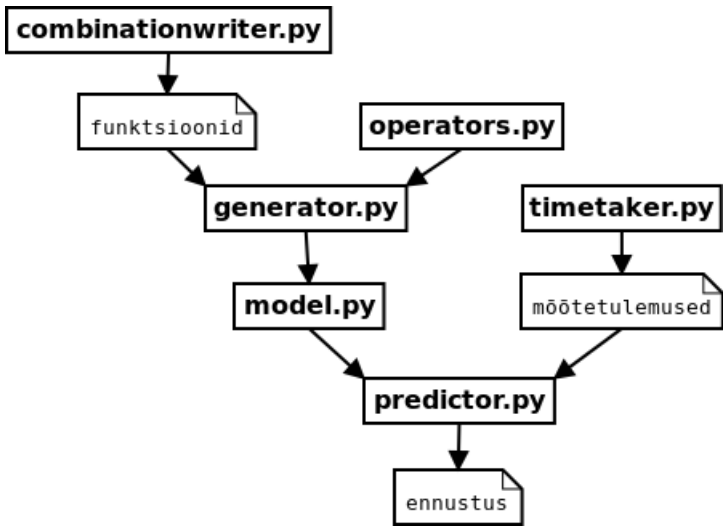
Sellise lähenemisega leitud mudel ei kannata potentsiaalselt ülesobituse käes, kuna lõpphinnang anti $D_{treening}$ andmetel treenitud mudelite testimisel mudeli mõistes tundmatutel D_{test} andmetel. Samuti on lahendatud alasobituse probleem, kuivõrd kõikide vaadeldud keerukusega ekvivalentsiklasside liikmetel on võimalik lõpuks valituks osutuda.

6 Implementatsioon

Ennustaja on kirjutatud programmeerimiskeeles Python versioonile 2.7 ning mõeldud kasutamiseks teegina. See tähendab, et kui meil on mingi programm, mille teatud osa soovime ennustada, saame importida ennustaja tööks tarvilikud moodulid ning neid kasutada kasutusjuhendis (osa 6.1) ette näidatud viisil. Genereeritud funktsioonide mugavaks parsimiseks on maksimaalne parameetrite arv piiratud kümneni.

Ennustaja kirjutamisel on kasutatud mittestandardset teeki mitmese lineaarse regressiooni probleemi lahendamiseks, kuivõrd tegu on arvutuslikult keerulise ülesandega. Regressioon on usaldatud Pythoni kolmanda osapoole teegile *pandas* [10], mis toetub C-keelsetele osadele ning mis võitis autori südame, kuivõrd meenutas oma andmestruktuuride poolest populaarse ja end tõestanud andmetöötluskeele R [11] Pythoni-keelset vastet. Valik on empiirilistel kaalutlustel osutunud õnnestunuks, kuna ennustaja profileerimisel selgus, et genereeritud mudelite arvule vastav kordi treeningandmetel jooksutatav regressioon võtab vaid ligikaudu 10% programmi tööajast.

Vajalikuks osutus ka *NumPy* [13] teek, kuivõrd *pandas* teisendas rohkem täpsust vajavad tulemused 64-bitisteks *NumPy* ujukomaarvudeks, millega rehkendamine vajas teisi samat tüüpi arve.



Joonis 2: Moodulite seos

Ennustaja on kirjutatud modulaarselt, millest kasutaja võiks kokku puu-

tuda vaid moodulitega *predictor.py*, *timetaker.py* ja *model.py*, mille kasutajale suunatud informatsioon on leitav dokumentatsioonist (lisa A). Moodulite seotust püüab kirjeldada joonis 6.

6.1 Kasutusjuhend

Käesolev kasutusjuhend katab lihtsate näidetega ära kõige olulisemad töö käigus valminud ennustaja funktsioonid. Lisaks tasub kasutajal tutvuda ka dokumentatsiooniga.

6.1.1 Installeerimine

Ennustaja vajab töötamiseks Pythoni versiooni 2.7 ning regressiooniks Pythoni kolmanda osapoole teeki *pandas*, mis on töö kirjutamise ajal kättesaadav aadressilt `http://pandas.pydata.org/getpandas.html`, ja *NumPy*, mille leiab aadressilt `http://sourceforge.net/projects/numpy/files/`.

Valminud programm koosneb lihtsatest *.py* formaati Pythoni moodulitest, mistõttu tuleb need importimisel kättesaadavaks teha. Selle hõlbustamiseks on nendest koostatud pakett, mis võimaldab ligipääsu kasutajale suunatud klassidele *Predictor* ja *TimeTaker*. Kasutaja saab läbi klassi *Predictor* juurdepääsu ka klassidele *Prediction* ja *Model*, kuid kasutajal on nende kättesaamine ette nähtud vaid tagastatava eksemplarina. Paketi kättesaadavaks tegemiseks on kaks erinevat viisi.

Esimeseks võimaluseks on kopeerida pakett (kaust) *pred* Pythoni *site-packages* kausta, kus resideeruvad kõik standardteegid ühes vareminstalleeritud pakettidega. Windowsi operatsioonisüsteemis on selle asukoht sageli "C:\Python26\Lib\site-packages". Linuxis võib selle asukoht olla "/usr/local/lib/python2.7/dist-packages". Kui Pythoni interpretaatori tee on lisatud operatsioonisüsteemi *path*'i, on mugav kasutada *site-packages* asukohta ütlevat käsku.

```
python -c "from distutils.sysconfig import get_python_lib;
print(get_python_lib())"
```

Teiseks võimaluseks on ajutiselt muuta Pythoni moodulite otsimise teed. Selleks tuleb lisada kaks rida enne ennustaja paketi *pred* importimist. Olgu ennustaja pakett *pred* kaustas "C:\kaustake".

```
import sys
sys.path.append("C:\\kaustake")
import pred
```

6.1.2 Aja mõõtmine

Kuivõrd tegu on masinõppe meetode rakendava lahendusega, on enne ennustamist tarvis mõõtmistulemusi, mille abil suudaks ennustaja luua vajaliku mudeli. Kujutagu meie programmi, mida soovime mõõta, lihtne töötamist simuleeriv funktsioon.

```
def program(n,m):
    result = 0
    for i in range(n):
        for j in range(m):
            result += i*j
```

Mõõtmist aitab läbi viia ennustaja klass *TimeTaker*. Selleks impordime *TimeTaker*'i

```
from pred import TimeTaker
```

ning loome uue eksemplari.

```
timetaker = TimeTaker()
```

Kui soovime, et mõõtmistulemusi ei salvestataks aktiivsesse kausta, lisame konstruktorile tee soovitud kaustani. Oletame, et soovime mõõtmistulemused salvestada aktiivse kausta alamkausta *time*.

```
timetaker = TimeTaker(directory="time")
```

Järgmisena seame *TimeTaker*'i mõõtma. Selleks peame kasutama unikaalset võtit, kuivõrd võtmenimelisse faili hakatakse salvestama vastava võtmega identifitseeritud mõõtmistulemusi. Olgu võtmeks "*program_time*".

Meil on vaja ette anda ka järjend arvuliste parameetritega, mis võivad tõenäoliselt mõõdetava programmi tööaega mõjutada. Kuivõrd mõõdetav programm *program* võtab kaks argumenti, on mõistlik arvata, et just need need sobivad antud juhul parameetriteks.

```
timetaker.start("program_time",[x,y])
program(x,y)
```

Pärast mõõdetava programmi peatumist on tarvis võtmega "*program_time*" määratud mõõtmine lõpetada.

```
timetaker.end("program_time")
```

Me saame korraga mõõta lõikuvaid programmiosasid seni, kuni paralleelselt mõõdetavate programmiosade võtmed ei kattu. Olgu meil mõõtmisel veel üks programm.

```
def program2(w)
    for k in range(w):
        print k
```

Oletame, et soovime korraga mõõta nii ainult *program* tööaega kui ka *program* ja *program2* tööaega koos. Tähistame viimase mõõtmise võtmega "*two_programs_time*". Sõltugu teine programm esimese programmi esimesest parameetrist.

```
timetaker.start("two_programs_time",[x,y])
timetaker.start("program_time",[x,y])
program(x,y)
timetaker.end("program_time")
program2(x)
timetaker.end("two_programs_time")
```

Nii *start* kui ka *end* on arvutuslikult kerged funktsioonid ilma failidesse kirjutamiseta, mistõttu pole kartust, et nende pesastamine rikuks oluliselt mõõtmistulemusi.

Peaks juhtuma, et meil on ühes *.py* failis mõõtmisel palju erinevaid programmilõike, kuid neist mõne aja soovime salvestada teistest erinevasse kausta, on meil see võimalus. Seame järgnevalt mõõtmise võtmega "*program_time*" salvestuma alamkausta *time2*.

```
timetaker.setSaveDir("program_time","time2")
```

Mõõtmiste lõppedes on kasulik mõõtmistulemused salvestada võtmetele vastavatesse failidesse. Selleks piisab ühest käsust, mis kirjutab senised mõõtmistulemused võtmete nimedega failidesse ning seejärel eemaldab tulemused mälust.

```
timetaker.publish()
```

Viimast käsku on mõistlik kutsuda siis, kui ühtegi mõõtmist parasjagu ei toimu, kuivõrd failioperatsioonid on ajakulukad ja võivad seega rikkuda mõõtmistulemusi.

6.1.3 Ennustamine

Ennustamiseks impordime klassi *Predictor*

```
from pred import Predictor
```

ning loome uue eksemplari.

```
predictor = Predictor()
```

Predictor'i konstruktor võimaldab ka konfigureerimist. Soovi korral on võimalik genereerimisele ja testimisele kulutada vaikimisi 5 sekundi asemel *duration*'i jagu sekundeid. *directory* laseb seada teed kaustani, kus on talletatud mõõtmistulemused – vaikimisi on selleks kaust, milles on ennustajat importinud programm. Argumendi *function_dir* kaudu on võimalik määrata kaust, kust lugeda või kuhu puudumisel ka kirjutada sõne kujul olevad funktsioonid, ning *function_symbols*'i abil saab määrata, mitme sümboli jagu liitmisi ja korrutatavaid tohib maksimaalselt esineda. Vaikimisi salvestatakse kuni 10 liitmise ja korrutatavaga valemid aktiivse kausta alamkausta "*functions*". Olgu soov lasta ennustajal kulutada mudeli otsingule 10 sekundit ning asugu andmed alamkaustas "*data*".

```
predictor = Predictor(duration=10,directory="data")
```

Olgu meil konstruktoris määratud kaustas võtmega "*program_time*" talletatud mõõtmistulemused, mis saadi, kui mõõtmisel anti kaheelemendiline järjend. Laseme ennustajal tagastada ennustuse objekti, andes järjendina ette parameetrid u ja v ning mõõtmistulemuste võtme.

```
prediction = predictor.predict([u,v],"program_time")
```

Ennustamisel on lisaks võimalik määrata ka *directory* näol ebastandardne kaust mõõtmistulemuste lugemiseks ning määr *in_testset* [0,1], kui suurt osa mõõtmistulemustest kasutada treeningandmestiku asemel valideerimisandmestikuna. Vaikimisi kasutatakse konstruktoris seatud kausta ning andmestik poolitatakse määra 0,5 abil. Oletame, et soovime eelneva koodi tingimustele lisaks seada ajutiseks mõõtmistulemuste kaustaks alamkausta "*new_data*" ning valideerimisadmestikku kõigest 20% mõõtmistulemustest. ‘

```
prediction = predictor.predict([u,v],"program_time",
                        directory="new_data",in_testset=0.2)
```

Ennustuse objektist annab kätte saada nii hinnatava aja

```
estimated_time = prediction.getEstimation()
```

kui ka mudeli, mis vaadeldavatest kirjeldas parameetrite kaudu kuluvat aega kõige paremini.

```
estimated_model = prediction.getModel()
```

Mudeli väljastamiseks valemina piisab selle väljaprintimisest.

```
print estimated_model
```

Kuivõrd parameetrite vaikenimed *par1* ja *par2* ei pruugi kasutajale valemis hästi öelda, kuidas aeg avaldub, saab neile enne printimist ka asjakohasemad nimed panna.

```
estimated_model.addParameterNames(["ridu","veerge"])
print estimated_model
```

7 Tulemused

Andmaks põhjendatud hinnangu implementeeritud ennustajale, testime seda reaalsel andmestikul ning vaatame ennustaja käitumist. Valitud andmestikuks on 1000 mõõtmist Linuxi *sort* programmil. Mõõtmistulemustes on kaks parameetrit: ridade arv n ning sümbolite arv reas m. Andmestiku loomisel on igal katsel valitud juhuslikult ridade arvuks 100-100000 ning sümbolite arvuks reas 100-20000.

Vaatame esmalt, kuidas sõltub ennustamise täpsus andmete hulgast. Laseme ennustajal iga kord genereerimise ja testimise peale kulutada 5 sekundit ning valime 10 korda 1000-st andmest juhuslikult alamandmestikud suurustega 20, 50, 500 ja 1000. Kümnest korrast igal korral valime nendest alamandmestikest välja 5 juhuslikku sissekannet ning leiame nende abil vastaval alamandmestikul ennustaja leitud mudeli keskmise vea. Seejärel leiame iga alamandestiku korral 10 katsekorra mudeli keskmise vea. Katse tulemust kujutab tabel 2.

Andmeid	**Keskmine ennustuse viga**
20	0,070 s
50	0,101 s
500	0,106 s
1000	0,108 s

Tabel 2: Keskmised ennustuse vead andmestiku mahu varieerudes

Ootuspärane oleks olnud vastupidine seos, kus vähemate andmete korral on keskmine ennustuse viga suurem, kuivõrd ennustajal on vähem informatsiooni. Kindlasti ei saa sellest võrdlemisi väheste juhuslikult valitud testide korral aga väita, justkui vähem andmeid tagaks parema ennustustulemuse. Katse alusel võib vaid kahtluse alla seada, kas rohkem andmeid tähendab nõnda lihtsa keerukusega (eeldatavasti $O(n \log(n)m)$) programmi korral tunduvalt paremat tulemust.

Järgnevalt vaatame, kuidas käitub ennustaja, kui varieerida genereerimiseks ja testimiseks lubatud aega. Olgu meil andmestikus 1000 mõõtmistulemust. Leiame iga aja korral parima mudeli ning leiame selle abil 30 juhusliku sissekande abil aja poolt määratud mudelile vastava keskmise ennustusvea. Katsetulemusi kajastab tabel 3.

Viimase katse tulemused on tunduvalt ootuspärasemad. On ilmne, et en-

Lubatud aeg	Keskmine ennustuse viga
0,1 s	4,410 s
1,0 s	0.864 s
2,0 s	0.168 s
5,0 s	0.111 s
10,0 s	0,105 s
15,0 s	0,107 s
20,0 s	0,120 s

Tabel 3: Keskmised ennustuse vead lubatud aja varieerudes

nustaja pole suutnud $0,1$ sekundiga genereerida ja testida piisavalt mudeleid, leidmaks aktsepteeritava täpsusega mudeli. Järgneva $0,9$ sekundi jooksul on täpsus märgatavalt paranenud ning juba 5 sekundist alates paistab olevat mingi etapi optimaalne mudel leitud. Ennustustäpsuse nõnda väikesed kõikumised alates 5 sekundist on tõenäoliselt vaid 30 juhuslikult valitud mõõtmiskatse mõju, kuivõrd programmide tööaegu mõõtes on müra vältimatu.

Analüüsime ka, kas ennustaja leiab parima võimaliku mudeli. Kuivõrd ootuspärane oleks, et aeg *sort*'il, mis on keerukusega $O(n\log(n)m)$, avalduks kõige paremini ligilähedase mudeli abil, leiame sellise mudelkuju korral andmetele kõige paremini vastavad koefitsiendid.

$$M(n,m) = 1,64 \times 10^{-9}(n\log(n)m) + 0,12$$

Sellisel juhul on ennustaja kohaselt valideerimisandmetel standardhälbe hinnanguks $\sim 0,14$. Seda ületab aga ootamatum mudel

$$M(n,m) = 1,82 \times 10^{-8}(nm) + 8,55 \times 10^{-20}(n^3m) + 3,85 \times 10^{-17},$$

mille standardhälbe hinnang valideerimisandmestikul on kõigest $\sim 0,04$. Veenmaks, et see võib tõesti ületada oodatut, on tehtud graafikud (leiab lisast B), kus on fikseeritud erinevates suurusjärkudes n ja m ning vaadatud, kuidas käituvad sellisel juhul eelnevad mudelid.

8 Summary

Predicting scientific computation's running time

Bachelor's thesis (6 ECTS)

Karl-Oskar Masing

The thesis consists of two major parts. In the first part, we present a technique to predict an arbitrary computation's running-time based on data gathered from previous executions. In the second part, we present an implementation along with a user manual and an example.

Running-time prediction is based on the following observation. We can distinguish between different program calls using parameters that potentially affect the running time. By recording both parameters and elapsed time, we can derive a model from the data that would estimate the running time. As we are interested in the comprehensibility of the model, we cannot use regular non-parametric regression methods. Hence, we use generalised linear regression with different basis functions such as n^2, $n \log n$ and $n^3 \log n$. However, as we approach the problem of finding a relatively simple yet accurate model while traversing the search space by generating different models, basis functions can get even more complicated, involving multiplications of multiple parameters with different powers. Using the found model, it is then possible to predict the running time when parameters are known. Thesis comes also with a Python library that uses the described method to estimate an arbitrary program's running-time.

Viited

[1] Additive model. http://en.wikipedia.org/wiki/Additive_model, aprill 2013.

[2] Bayesian information criterion. http://en.wikipedia.org/wiki/Bayesian_information_criterion, mai 2013.

[3] Decision tree. http://en.wikipedia.org/wiki/Decision_tree, märts 2013.

[4] Eureqa. http://creativemachines.cornell.edu/eureqa, mai 2013.

[5] Latex. http://www.latex-project.org/, mai 2013.

[6] Latex. http://dev.midnightcoding.org/projects/gummi, mai 2013.

[7] Nearest neighbor search. http://en.wikipedia.org/wiki/Nearest_neighbor_search, märts 2013.

[8] Neural network. http://en.wikipedia.org/wiki/Neural_network, märts 2013.

[9] Nonparametric regression. http://en.wikipedia.org/wiki/Nonparametric_regression, aprill 2013.

[10] Python data analysis library - pandas. http://pandas.pydata.org, mai 2013.

[11] The r project for statistical computing. http://www.r-project.org, mai 2013.

[12] Regression analysis. http://en.wikipedia.org/wiki/Regression_analysis, aprill 2013.

[13] Scientific computing tools for python. `http://www.numpy.org`, mai 2013.

[14] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[15] Steven M. Bellovin. The security flag in the ipv4 header. Request for Comments 3514, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc3514.txt, 2003. aprillinali.

[16] Corinna Cortes, Lawrence D. Jackel, and Wan-Ping Chiang. Limits on learning machine accuracy imposed by data quality. In *KDD'95*, pages 57–62, 1995.

[17] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, märts 2009.

[18] Deirdre M. Kirke. Gender clustering in friendship networks: some sociological implications. *Methodological Innovations Online*, 4(1):23–36, 2009.

[19] Tom M. Mitchell. The discipline of machine learning. Professori nägemus, juuli 2006.

[20] Jay I. Myung, Daniel J. Navarro, and Mark A. Pitt. Model selection by normalized maximum likelihood, 2005.

[21] Valerie Sessions and Marco Valtorta. The effects of data quality on machine learning algorithms. In *ICIQ'06*, pages 485–498, 2006.

[22] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times with historical information. *J. Parallel Distrib. Comput.*, 64(9):1007–1016, September 2004.

[23] Alex Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*, page 66. Cambridge University Press, 1st edition, 2008.

[24] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Pearson Higher Ed USA, 1st edition, 2005.

Lisad

A Dokumentatsioon

A.1 Prediction

Kirjeldus:	hoiab ennustuse tulemusi
Moodul:	predictor.py

Meetodid

Signatuur:	getEstimation()
Kirjeldus:	tagastab ennustustatud aja sekundites
Tagastustüüp:	ujukomaarv

Signatuur:	getModel()
Kirjeldus:	tagastab ennustamisel kasutatud mudeli
Tagastustüüp:	Model

Signatuur:	getParameters()
Kirjeldus:	tagastab ennsutamisel kasutatud parameetrid
Tagastustüüp:	arvude järjend

A.2 Predictor

Kirjeldus:	vastutab ennustamise eest
Moodul:	predictor.py

Konstruktor

Predictor([duration [, directory [, function_dir [, function_symbols]]]])

Parameeter:	**Kirjeldus:**	**Tüüp**
duration	genereerimisele ja testimisele kulutatav aeg sekundites	arv
directory	tee mõõtmistulemusteni	sõne
function_dir	tee, kust lugeda/talletada genereeritud funktsioone	sõne
function_symbols	sümbolite arv funktsioonis	täisarv

Meetodid

Signatuur:	predict(parameters, keyname[, directory[, in_testset]])
Kirjeldus:	tagastab ennustustatud aja sekundites
Parameetrid:	parameters – ennustamisel kasutatavad parameetrid tüüp: arvude järjend keyname – mõõtmistulemuse identifikaator tüüp: sõne directory – tee mõõtmistulemuste kaustani tüüp: sõne vaikimisi: “” in_testset – valideerimisandmestikku kuulumise määr tüüp: ujukomaarv [0,1] vaikimisi: 0.5
Tagastustüüp:	Prediction

A.3 Model

Kirjeldus:	kujutab ennustamisel kasutatud mudelit
Moodul:	model.py

Meetodid

Signatuur:	estimate(values[,min_est])
Kirjeldus:	tagastab mudeli väärtuse kohal *values*
Parameetrid:	values – mudelile rakendatavad väärtused tüüp: arvude järjend min_est – minimaalne tagastatav väärtus tüüp: ujukomaarv vaikimisi: 0.00001
Tagastustüüp:	ujukomaarv

Signatuur:	addParameterNames(parameter_names)
Kirjeldus:	asendab mudeli kuvamisel parameetrid vastavate nimedega
Parameetrid:	parameter_names – parameetrite nimed tüüp: sõne järjend
Tagastustüüp:	-

A.4 TimeTaker

Kirjeldus:	vastutab programmide tööaja mõõtmise eest
Moodul:	timetaker.py

Konstruktor

TimeTaker([directory])

Parameeter:	Kirjeldus:	Tüüp
directory	tee salvestatavate mõõtmistulemuste kaustani	sõne

Meetodid

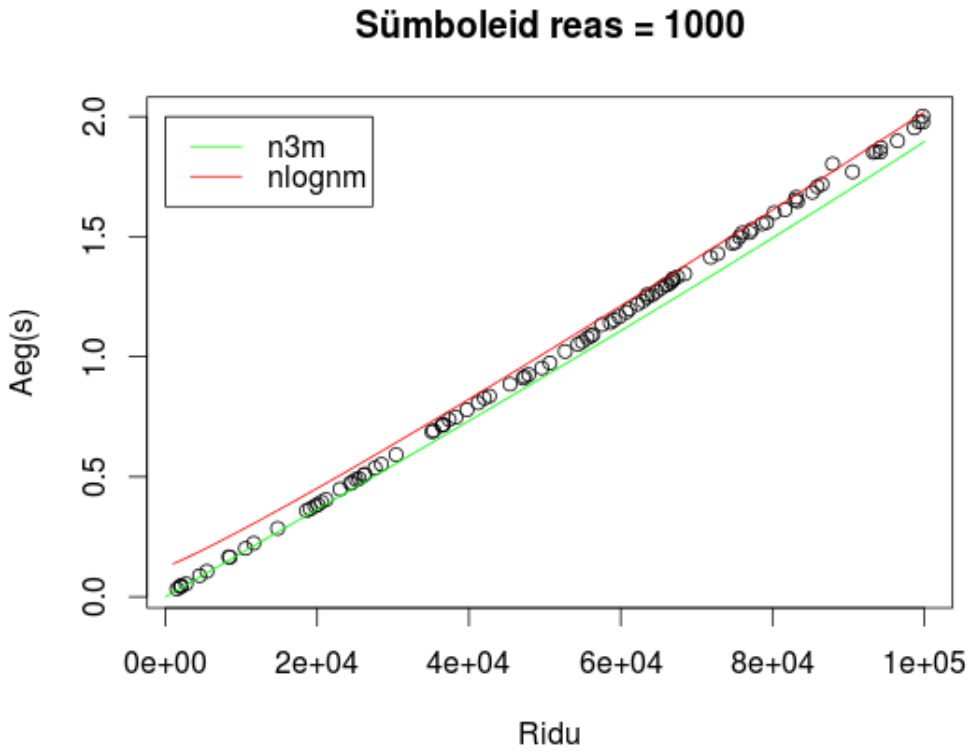
Signatuur:	start(keyname,parameter_list)
Kirjeldus:	alustab mõõtmist võtmega *keyname*
Parameetrid:	keyname – mõõtmise võti tüüp: sõne parameter_list – parameetrid, millest mõõtmistulemus võib sõltuda tüüp: arvude järjend
Tagastustüüp:	-

Signatuur:	end(keyname)
Kirjeldus:	lõpetab mõõtmise *keyname* ja salvestab tulemuse mällu
Parameetrid:	keyname – mõõtmise võti tüüp: sõne
Tagastustüüp:	-

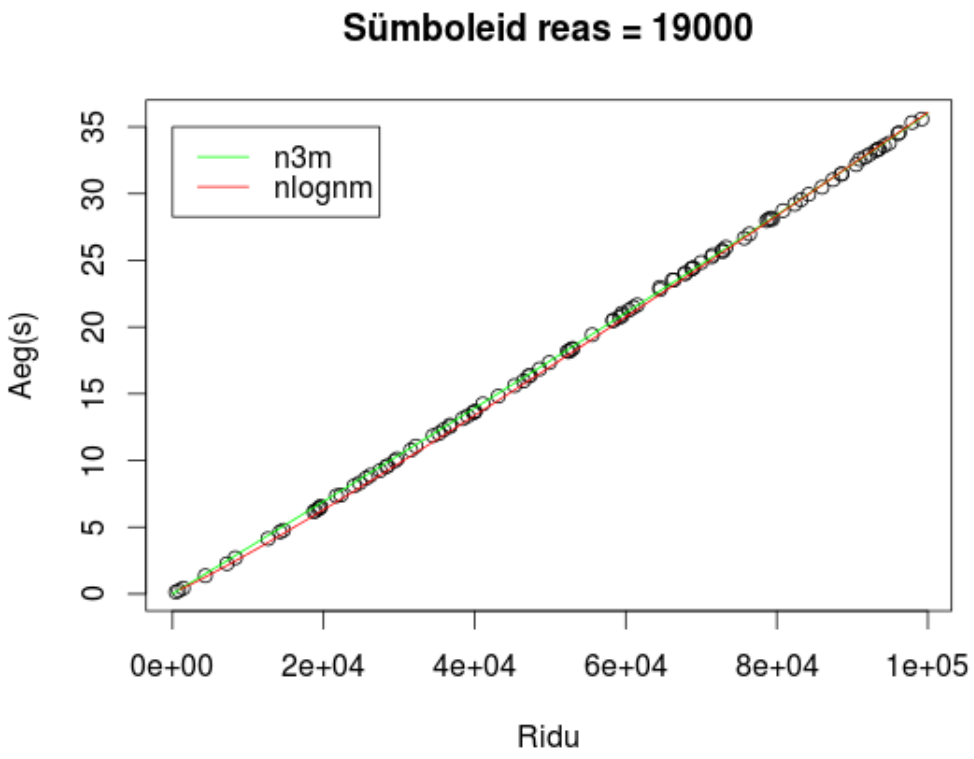
Signatuur:	publish()
Kirjeldus:	kirjutab mõõtmistulemused vastava võtme nimega failidesse
Tagastustüüp:	-

Signatuur:	setSaveDir(keyname,destination)
Kirjeldus:	seab mõõtmise *keyname* salvestuma kausta teega *destination*
Parameetrid:	keyname – mõõtmise võti tüüp: sõne destination – tee uue salvestamise kaustani tüüp: sõne
Tagastustüüp:	-

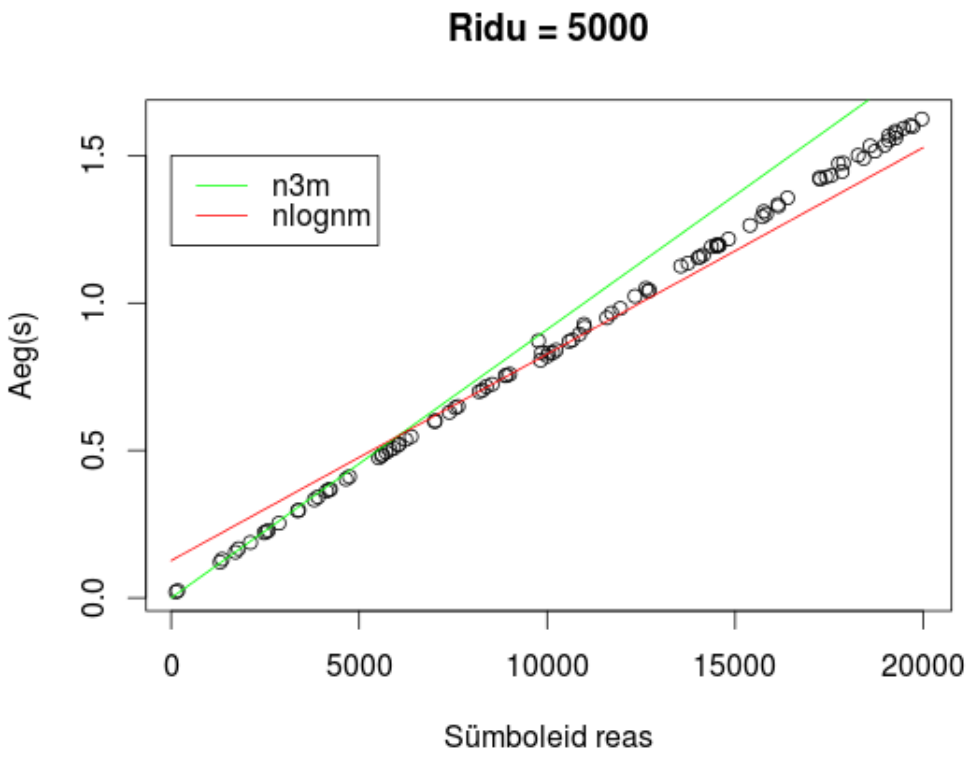
B Graafikud



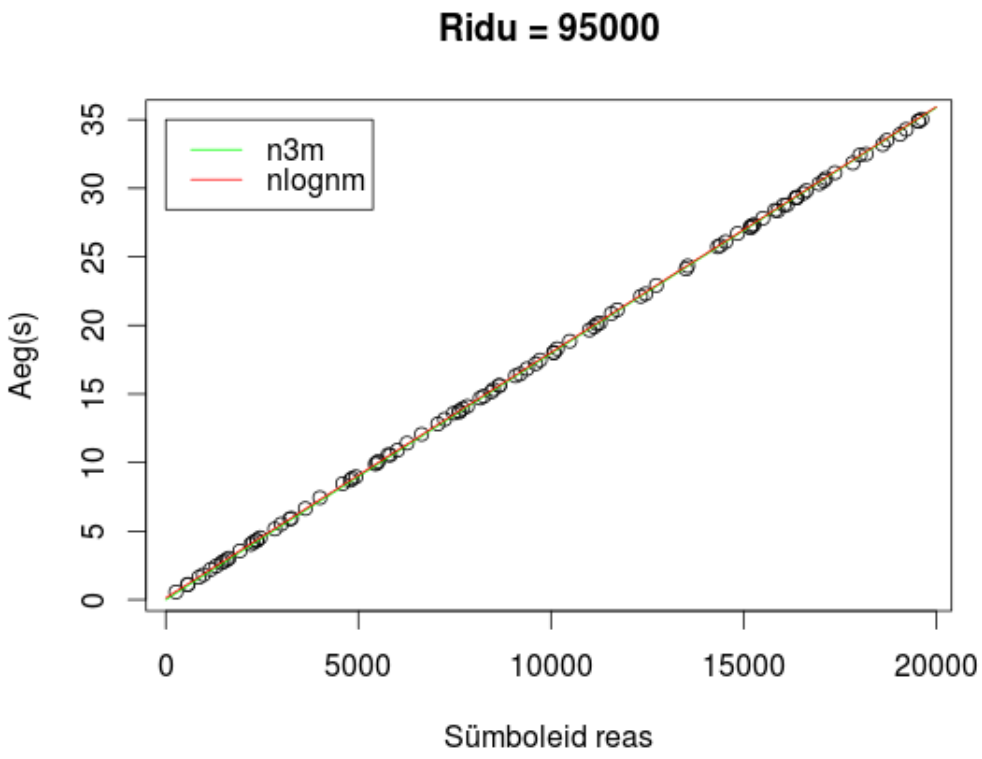
Joonis 3: n^3m mudel vs $n\log(n)m$



Joonis 4: n^3m mudel vs $n\log(n)m$



Joonis 5: n^3m mudel vs $n\log(n)m$



Joonis 6: n^3m mudel vs $n\log(n)m$

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Karl-Oskar Maring
(*autori nimi*)
(sünnikuupäev: 14.10.1989)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„Teadusarvutuse tööaja ennustaja"
(*lõputöö pealkiri*)
mille juhendajad on dr Meelis Kull ja dr Sven Laur,
(*juhendajate nimed*)

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, **13.05.2013**