TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

Sten Uduste 121053IAPB

SISSETUNGI AVASTAMISE ANDMEKOGUMI KLASTERDAMINE KASUTADES K-KESKMISTE MEETODIT

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc
Lektor

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Sten Uduste

03.01.2017

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on anda ülevaade sissetungi avastamise andmekogumi klasterdamisest, kasutades selleks k-keskmiste meetodit. Lisaks sellele on välja toodud erinevaid klasterdamise optimeerimise võimalusi ning nende rakendamisel saadud tulemused.

Töös toob autor välja erinevaid meetodeid k-keskmiste klasterdamise ning selle kiiruse parema tulemuse saavutamiseks. Klasterdamise ja optimeerimise meetodite analüüsimiseks on kasutatud sissetungi avastamise andmekogumit NSL-KDD.

Välja toodud k-keskmiste klasterdamise optimeerimise meetodite kombinatsioonina saavutati ligi 30% võrra parem üleüldine klasterdamise täpsus ning selleks kulunud aja ligi 10 korra kiirenemine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 6 peatükki, 2 joonist, 11 tabelit.

Abstract

Intrusion detection data set clustering using k-means method

The aim of this Bachelor's thesis is to give an overview of intrusion detection data set clustering using k-means method. In addition to clustering various optimization options and their implementation results are presented.

In this study the author brings out various methods in order to achieve faster clustering and its results. Intrusion detection data set NSL-KDD is used to analyze the clustering and its optimization methods.

Using all of the optimization methods presented in this study a nearly 30% increase in overall clustering result was achived and the time used for it became nearly 10 times faster.

The thesis is in Estonian and contains 28 pages of text, 6 chapters, 2 figures, 11 tables.

Lühendite ja mõistete sõnastik

DoS	DoS rünnak (ingl. k. *Denial of service*), kus ründaja üritab takistada õigustatud kasutajatel teenuse kasutamist [10, lk 2].
Probe	Probe rünnak (ingl. k. *Probe*), kus ründaja proovib koguda andmeid arvutite võrgustiku kohta, et leida nende turvalisuse nõrki kohti, mida kasutada sissetungimiseks [10, lk 2].
R2L	R2L rünnak (ingl. k. *Remote to User*), kus ründajal on võimalus saata süsteemile pakette läbi võrgu ning proovib selle abil saada kohaliku juurdepääsu süsteemile [10, lk 2].
U2R	U2R rünnak (ingl. k. *User to Root*), kus ründajal on kohalik juurdepääs ohvri süsteemile ja üritab saada süsteemi administratiivse kasutaja õiguseid [10, lk 2].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval omavad paljud inimesed arvutit ning hoiavad seal tähtsaid ja tundlikke andmeid. Loomulikult tahame, et need andmed ei satuks valedesse kätesse. Selle tagamiseks aitavad kaasa sissetungi avastamise süsteemid, mis jälgivad interneti või süsteemide võrgustikus toimunud tegevusi, analüüsides neid ning annavad märku pahatahtlike tegevuste leidmisel. Kuna jälgitavaid andmeid on suurtes kogustes, siis iseenesestmõistetavalt võtab nende analüüsimine palju aega. Sellest tulenevalt on sissetungi avastamise süsteemides kasutusele võetud erinevaid andmekaeve tehnikaid, mis aitavad andmekogusid struktureerida ning selle tulemusena parandada süsteemide kiirust ja nende tulemusi.

1.1 Eesmärk

Antud töö eesmärgiks on anda ülevaade sissetungi avastamise andmekogumi normaalsete ja rünnaku ühenduste klasterdamisest, kasutades selleks k-keskmiste klasterdamist. Lisaks sellele on eesmärgiks parandada klasterdamiseks kuluvat aega ning samaaegselt saavutada võimalikult hea grupeerimise täpsus.

1.2 Metoodika

Antud töö eesmärkide saavutamiseks on autor uurinud erinevaid teatmekirjandusi leidmaks võimalikke meetodeid k-keskmiste klasterdamise parema tulemuse saavutamiseks. Kasutatud meetodite ja klasterdamise tulemuste analüüsimiseks on kasutatud sissetungi avastamise andmekogumit NSL-KDD.

1.3 Ülevaade tööst

„K-keskmiste klasterdamine“ peatükis on antud ülevaade k-keskmiste klasterdamisest ning töös kasutatud klasterdamise algoritmidest. Järgnevas peatükis „Andmekogum“ on kirjeldatud töös kasutatavat andmekogumit. Peatükis „Klasterdamise efektiivsuse hindamine“ on kirjeldatud töös kasutatavate klasterdamiste efektiivsuse hindamise

meetodit ja selleks vajalikke valemeid. Töö kõige mahukamas peatükis „K-keskmiste klasterdamise optimeerimine“ on välja toodud erinevaid meetodeid k-keskmiste klasterdamise parema tulemuse saavutamiseks ning nende realiseerimisel saadud tulemused.

2 K-keskmiste klasterdamine

Klasterdamine on andmekaeve tehnika, mis tegeleb uuritavate andmete grupeerimisega klastritesse. Klasterdamise tehnikad kuuluvad üldjuhul järelvalveta tehnikate hulka, nimelt ei tegeleta sihikindlate muutujate avastamisega, vaid proovitakse leida struktuure terve andmekogumi peale. Grupeerimine toimub ainult andmete atribuutide baasil, mille ülesandeks on leida üles üksteisega kuidagimoodi sarnased andmed ning panna need ühte kobarasse, klastrisse. Klasterdatud andmed peaksid olema klastrisiseste andmetega sarnased ja samas teistesse klastrisse kuuluvate andmetega võimalikult erinevad [12].

K-keskmiste klasterdamine on üks vanim (idee on välja mõeldud Hugo Steinhausi poolt 1957 aastal [8])ja enim levinum klasterdamise tehnika. Tehnika populaarsus tuleneb selle rakendamise lihtsusest ja kiirusest. K-keskmiste klasterdamise ülesandeks on andmekogumik grupeerida eelnevalt määratud arv kobaratesse ehk klastritesse. Klasterdamine toimub uuritavate andmete ja olemasolevate klastrite kauguse suhte leidmisel, kus anne määratakse temast lähimasse klastrisse. Tulemuseks on võimalikult minimaalne klastri ja klastrisse kuuluvate andmete erinevus ehk kaugus. Klastreid vaadatakse kui nendesse kuuluvate andmete aritmeetilist keskmist ning kauguse leidmisel võrreldakse just seda keskpunkti ja uuritavat annet. Sellest tulenevalt on k-keskmiste klasterdamine keskpunkti baasiline, kus klastri keskpunkt esindab kõiki sellesse klastrisse kuuluvaid andmeid. Kauguse leidmiseks on tavaliselt ning samuti ka antud töös kasutatud eukleidilist kaugust (1), kus f on atribuutide arv ning C_i ja X_i on klastri ja objekti vastava atribuudi väärtus [12].

$$D(c,x) = \sqrt{\sum_{i=1}^{f}(c_i - x_i)^2} \quad (1)$$

Kuna k-keskmiste meetod on suhteliselt lihtne siis on sellel ka nõrkki külgi, millest suurimateks on parima klastrite arvu määramine, esialgsete klastrite valimine ning võõrväärtuste (ingl. k. outliers) olemasolu. Võõrväärtused on teistest suurelt erinevate objektid, atribuudid [12].

2.1 Lloyd ja MacQueen k-keskmiste klasterdamise algoritmid

Antud alampeatükis on kirjeldatud töös kasutatud Lloydi ja MacQueeni k-keskmiste klasterdamise algoritme.

Nagu eelnevalt mainitud on k-keskmiste klasterdamine suhteliselt vana tehnika. Lloyd k-keskmiste klasterdamise algoritm on Stuart Lloydi poolt 1957. aastal [15] välja pakutud tehnika, mis oli esialgselt mõeldud uuritud analoog signaalide digitaalseks esindamiseks, millest on nüüdseks saanud k-keskmiste klasterdamise standard algoritmiks. Välja pakutud tehnikat ei avalikustatud aga väljaspool Bell Labsi kuni 1982. aastani. Terminit k-keskmiste kasutati esmakordselt James MacQueeni poolt 1967. aastal [6],[8].

Lloydi k-keskmiste klasterdamise algab, vastavalt eelnevalt määratud kobarate arvule, klastrite esialgse keskpunktide valimisega. Vaikimisi valitakse need suvaliselt uuritavast andmekogumist. Järgnevalt määratakse uuritavad andmed lähimasse klastrisse, kasutades selleks klastri ja andme vahelist kaugust. Seejärel pärast kõigi andmete määramist esialgsetesse gruppidesse uuendatakse klastrite keskpunkte. Järgnevalt kontrollitakse andmete ja uute kobarate vahelist kaugust ning uue lähima klastri korral liigutatakse andmeid klastrist klastrisse. Pärast kõigi andmete liigutamist uuendatakse järjekordselt gruppide keskpunkte. Viimast tegevust korratakse senikaua, kuni ükski anne enam gruppe ei vaheta või on saavutatud selle tegevuse ehk iteratsiooni maksimaalne arv. Järgnevalt on formaalselt kirjeldatud Lloyd k-keskmiste klasterdamise algoritmi [12]:

1. Valitakse k suvalised objektid klastrite $C_1,C_2,...,C_k$ esialgseteks keskpunktideks.

2. Iga objekt $X_1,...,X_n$ määratakse lähimasse klastrisse, kus objekti ja klastri keskpunkti vaheline kaugus on minimaalne.

3. Uuendatakse kõigi klastrite keskpunkte.

4. Sammu 2 ja 3 korratakse niikaua, kuni ükski objekt enam klasterid ei vaheta või on saavutatud iteratsioonide maksimaalne arv.

MacQueeni algoritmi erinevus Lloydi omaga on klastrite keskpunkti uuendamise protsess. Lloydi k-keskmiste klasterdamisel uuendatakse kobarate keskpunkte pärast kõigi objektide määramist või liigutamist klastrisse ning keskpunktide muutmiseks peab kogu andmekogumi klastrite arvu korra läbi käima. MacQueeni k-keskmiste klasterdamisel uuendatakse klastrite keskpunkte koheselt pärast klastri muutumist ning selleks ei pea läbi käima tervet andmekogumit [9, lk 3]. Sellest tulenevalt on MacQueeni algoritm tunduvalt kiirem ning keskpunktide arvutamiseks saab kasutada valemeid (2) ja (3) [5]. Valemit (2) kasutatakse uue objekti a_n lisandumisel klastrisse, kus *s* on kobara eelnev aritmeetiline keskmine. Valemit (3) kasutatakse, kui klastrist eemaldatakse objekt a_n,kus *s* on grupi eelnev aritmeetiline keskmine. Mõlemas valemis on *n* objektide arv klastris. Lloydi ja MacQueeni klasterdamine on välja toodud antud töö eesmärkide saavutamiseks loodud Java programmist lisas 1 ja 2.

$$\dot{s} = s + \frac{a_{n+1} - s}{n+1} \tag{2}$$

$$\ddot{s} = \frac{n * s - a_n}{n-1} \tag{3}$$

3 Andmekogum

KDD99 (ingl. k. Knowledge Discovery in Databases) [7] on laialt levinud andmekogum, mis on üles ehitatud 1998. aastal DARPA [2] algatusel pakkuda sissetungi avastamise süsteemidel andmekogumit, mille põhjal saaks hinnata ja võrrelda erinevate meetodite efektiivsust. KDD99 andmekogum on aga aegunud ja seoses tehnoloogia arenguga ei suuda enam täpseid hindamisi pakkuda, kuid avalikke alternatiivsete andmekogumite puudumise tõttu on see siiski laialt kasutuses. Antud töös on kasutatud NSL-KDD [11] andmekogumit, mis on KDD99 uuendatud versioon, kus on parandatud mõningaid andmekogumile omaseid probleemid [10].

Andmekogum koosneb interneti ühenduste kirjetest, mis on liigitatud normaalseks või rünnakuks. Rünnakutena on kasutatud nelja põhilist rünnakute liiki [10, lk 2]:

- DoS (ingl. k. *Denial of Service*) – Ründaja üritab takistada õigustatud kasutajatel teenuse kasutamisest.
- Probe (ingl. k. *Probe*) – Ründaja proovib koguda andmeid arvutite võrgustiku kohta, et leida nende turvalisuse nõrkki kohti, mida kasutada sissetungimiseks.
- U2R (ingl. k. *User to Root*) – Ründajal on kohalik juurdepääs ohvri süsteemile ja üritab saada süsteemi administratiivse kasutaja õiguseid.
- R2L (ingl. k. *Remote to Local*) – Ründajal on võimalus saata süsteemile pakette läbi võrgu ning proovib selle abil saada kohaliku juurdepääsu süsteemile.

NSL-KDD andmekogumit on kahte tüüpi treening ja testimine. Treeningu andmekogumit kasutatakse uuritava rünnaku tabamise meetodi arendamiseks ning testimise andmekogumit arendatud meetodi tõhususe hindamiseks. Treening ja testimise andmekogumis kasutatud rünnakute tüübid on välja toodud tabelis 1, kus treening andmekogum sisaldab 21 erinevat rünnaku tüüpi ning testimise andmekogumis on lisaks veel kasutusel 18 uut rünnaku tüüpi. Tabelis 1 on testimise andmekogumis lisandunud rünnakute tüübid märgistatud tärniga (*). Mõlema andmekogumi kirjete arv,

rünnaku liikide koguarv ning protsentuaalne sisaldatavus on välja toodud tabelis 2. Lisaks on tabelis 3 välja toodud andmekogus sisalduvate andmetele kuuluvad 41 omaduste nimetused (inglise keeles) koos nende järjekorra numbriga ning nende väärtuste tüübid.

Tabel 1. Rünnakute liigid ja nende tüübid.

Rünnaku klass	**Rünnaku tüüp**
DoS	Land, Neptune, Pod, Smurf, Teardrop, Back, Apache2*, Udpstorm*, Processtable*, Worm*, Mailbomb*
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan*, Saint*
R2L	Guess_passwd, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock*, Xsnoop*, Snmpguess*, Snmpgetattack*, Httptunnel*, Sendmail*, Named*
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl*, Sqlattack*, Xterm*, Ps*

Tabel 2. NSL-KDD andmekogumi kirjete arv.

Andmekogumi tüüp	**Koguarv**					
	Andmeid	**Normal**	**DoS**	**Probe**	**U2R**	**R2L**
Treening	125973	67343	45927	11656	52	995
		53,46%	36,46%	9,25%	0,04%	0,79%
Test	22544	9711	7460	2421	67	2885
		43,08%	33,09%	10,74%	0,30%	12,80%

Tabel 3. Ühenduste omadused ja nende väärtuste tüübid.

Väärtuse tüüp	**Omaduse nimetus (järjekorra number)**
Nominaalne	protocol_type (2), service (3), flag (4)
Binaarne	land (7), logged_in (12), root_shell (14), su_attempted (15), is_host_login (21), is_guest_login (22)
Protsentuaalne	serror_rate (25), srv_serror_rate (26), rerror_rate (27), srv_serror_rate(28), same_srv_rate (29), diff_srv_rate (30), srv_diff_host_rate (31), dst_host_same_srv_rate (34), dst_host_diff_srv_rate (35), dst_host_same_src_port_rate (36), dst_host_srv_diff_host_rate (37), dst_host_serror_rate (38), dst_host_srv_serror_rate (39), dst_host_rerror_rate (40), dst_host_srv_rerror_rate (41)
Arvuline	duration (1), src_bytes (5), dst_bytes (6), wrong_fragment (8), urgent (9), hot (10), num_failed_logins (11), num_compromised (13), num_root (16),

Väärtuse tüüp	**Omaduse nimetus (järjekorra number)**
	num_file_creations (17), num_shells (18), num_access_files (19), num_outbound_cmds (20), count (23), srv_count (24), dst_host_count (32), dst_host_srv_count (33)

4 Klasterdamise efektiivsuse hindamine

Antud töös on k-keskmiste klasterdamise eesmärgiks grupeerida rünnakute ühendused ja normaalsed ühendused eraldi klastritesse. Klasterdamisel saadud klastrite klassifitseerimiseks on kasutatud andmekogumis määratud ühenduste nimetusi. Kui klastris on rohkem rünnakute ühendusi, siis on tegemist rünnakute klastriga, vastasel juhul normaalsete ühenduste klastriga.

Klasterdamise algoritmi efektiivsuse hindamiseks on kasutatud selle kiirust ja klassifitseerimise ehk grupeerimise täpsust (4, 5, 6) (edaspidi täpsus) ning valepositiivsete (7) ja -negatiivsete (8) tulemuste määra. Nende leidmiseks on vaja teada järgnevaid arve [13]:

- ÕP (õigepositiivne) – Rünnaku ühenduste arv, mis on klassifitseeritud kui rünnaku ühendus.
- ÕN (õigenegatiivne) – Normaalsete ühenduste arv, mis on klassifitseeritud kui normaalne ühendus.
- VP (valepositiivne) – Normaalsete ühenduste arv, mis on klassifitseeritud kui rünnaku ühendus.
- VN (valenegatiivne) – Rünnaku ühenduste arv, mis on klassifitseeritud kui normaalne ühendus.

$$CR_{Total} = \frac{\tilde{O}N+\tilde{O}P}{\tilde{O}N+VP+VN+\tilde{O}P} \quad (4)$$

$$CR_{normal} = \frac{\tilde{O}N}{\tilde{O}N+VP} \quad (5)$$

$$CR_{rünnak} = \frac{\tilde{O}P_{rünnak}}{\tilde{O}P_{rünnak}+VN_{rünnak}} \quad (6)$$

$$VPM = \frac{VP}{VP+\tilde{O}N} \quad (7)$$

$$VNM = \frac{VN}{VN+\tilde{O}P} \quad (8)$$

5 K-keskmiste klasterdamise optimeerimine

Järgnevates alampeatükkides toob autor välja teatmekirjandustes pakutud meetodeid k-keskmiste klasterdamise parema tulemuse saavutamiseks. Välja toodud meetodid analüüsitakse peatükis 3 kirjeldatud NSL-KDD andmekogumi põhjal, kasutades klasterdamise ja meetodite realiseerimiseks Java programmeerimiskeelt.

5.1 Testkeskkond

- Protsessor Intel i7-4700MQ 2,40GHz
- 16GB DDR3 mälu
- Java versioon 1.8.0_92

Kuna mõlemad töös uuritavad k-keskmiste klasterdamise meetodid valivad esialgseteks klastrite keskpunktideks suvalised andmed, siis ei ole tagastatavad tulemused alati samasugused. Sellest tulenevalt on töös uuritud meetodeid korratud 100 korda ning tulemusteks toodud nende aritmeetiline keskmine. Tabelites välja toodud klasterdamiseks kulunud aega on mõõdetud sekundites ning iteratsioonide arvu tähisteks on *R*.

5.2 Klastrite arvu valimine

K-keskmiste klasterdamise jaoks on esmalt vaja määrata klastrite arv. Kuna erinevate klastrite arvu korral on tulemused erinevad, oleks vaja leida klastrite arv, mille tulemused oleksid parimad. Ükshaaval erinevate klastrite korral saadud tulemuste uurimine osutub aga tülikaks. Sellepärast on antud töös kasutatud küünarnuki meetodit sobiva klastrite arvu leidmiseks.

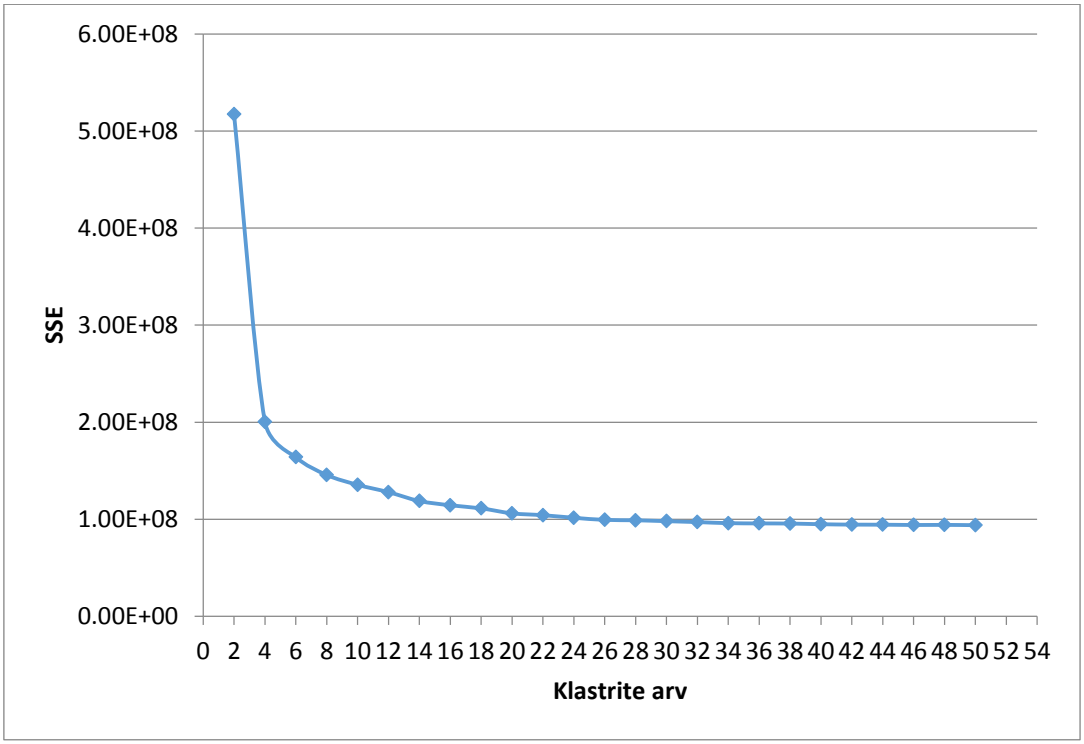
Küünarnuki meetod on naiivne visuaalne meetod leidmaks sobiliku klastrite arv. Antud meetodis leitakse erinevate klastrite arvu kasutamisel saadud klastrite sisese kauguste ruudu summad (ingl. k. *sum of the squared errors*) (edaspidiselt SSE) (9). Saadud

tulemused kantakse graafikule klastrite suurenemise järjekorras ning leitakse sealt „küünarnukk“. Küünarnuki all on mõeldud graafikul esinevat nn. nõksu, kus SSE muutumine, klastrite arvu suurenemisel, muutub ühtlaselt minimaalseks [12, lk 60].

$$SSE = \sum_{i=1}^{k} \sum_{X \in C_i} D(C_i, X)^2 \quad (9)$$

Küünarnuki meetodi, kasutades klasterdamiseks Lloyd meetodit, tulemus on välja toodud joonisel 1, kus on näha, et pärast klastrite arvu 22 muutub SSE suhteliselt minimaalselt aina väiksemaks. Kontrollimaks, et klastrite arvu 22 korral saadud tulemused on rahuldavad, on erinevate klastrite arvu korral saadud klasterdamise tulemused välja toodud tabelis 4 ja 5. Antud tulemustest on näha,et klastrite arvu 10 ja 15 korral on rünnakute klasterdamine väga minimaalne. Klastrite arvu 30 ja 50 korral on klasterdamise üleüldine täpsus parem, aga klasterdamise ajakulu ja rünnakute grupeerimine on tunduvalt halvem.

Edasisteks analüüsimisteks on klastrite arvuks valitud 22, mis on antud töö eesmärkide saavutmiseks andnud parima tulemuse.



Joonis 1. Lloyd klasterdamise SSE tulemused erinevate klastrite arvu korral.

Tabel 4. MacQueen klasterdamise meetodi tulemused erinevate klastrite arvu korral.

K	CR_{Normal}	CR_{DoS}	CR_{Probe}	CR_{U2R}	CR_{R2L}	Aeg	R	CR_{Total}
10	99,57%	1,45%	2,31%	0,73%	1,93%	0,87	18,53	53,99%
15	99,62%	2,04%	5,01%	0,00%	1,51%	1,88	26,89	54,48%
22	33,48%	95,12%	97,69%	35,08%	58,24%	4,36	43,78	62,09%
30	52,84%	95,12%	97,69%	17,31%	55,62%	14,26	108,34	62,09%
50	90,68%	97,58%	73,24%	10,60%	37,40%	42,41	190,17	91,12%

Tabel 5. Lloyd klasterdamise meetodi tulemused erinevate klastrite arvu korral.

K	CR_{Normal}	CR_{DoS}	CR_{Probe}	CR_{U2R}	CR_{R2L}	Aeg	R	CR_{Total}
10	99,88%	0,02%	4,33%	0,00%	2,23%	1,81	30,49	53,81%
15	99,57%	2,05%	5,26%	0,00%	1,51%	3,88	42,38	54,48%
22	18,25%	98,95%	97,65%	68,08%	80,79%	15,57	120,04	55,52%
30	48,16%	95,12%	97,64%	22,58%	55,67%	32,83	190,45	69,91%
50	94,55%	87,62%	51,38%	4,08%	13,74%	145,65	534,10	87,34%

5.3 Ühenduste omaduste valimine

Järgnevas alampeatükis refereerin *Hybrid Feature Selection for Network Intrusion* [14] tööd, v.a. seal, kus on märgitud teisiti.

Antud töö andmekogumites on igal andmel 41 atribuuti, omadust mille põhjal andmeid klasterdatakse. Rohkete atribuutide arvu korral võivad aga mõned atribuudid klasterdamisel vähem rolli mängida ning samas võivad normaliseeritud kujul erinevad atribuudid üksteisega korreleeruda, mis mõjutab klasterdamise tulemust [1]. Selleks, et vähendada klasterdamisel kasutatavaid ühenduse omadusi on antud töös leitud kõige enam rolli mängivad atribuudid.

Tähtsaimate omaduste valimiseks on leitud omaduste informatsiooni kasu (ingl. k. information gain), mille leidmine näeb välja järgnevalt. Olgu meil m arv klasse ja S_i arv kirjeid, mis kuuluvad klassi i ning kus S on kirjete koguarv. Nõutav informatsiooni entroopia kirjete klassifitseerimiseks leitakse kasutades valemit (10) [14, lk 3]

$$E(S_1, S_2, \dots, S_m) = -\sum_{i=1}^{m} \frac{S_i}{S} log_2 \frac{S_i}{S} \quad (10)$$

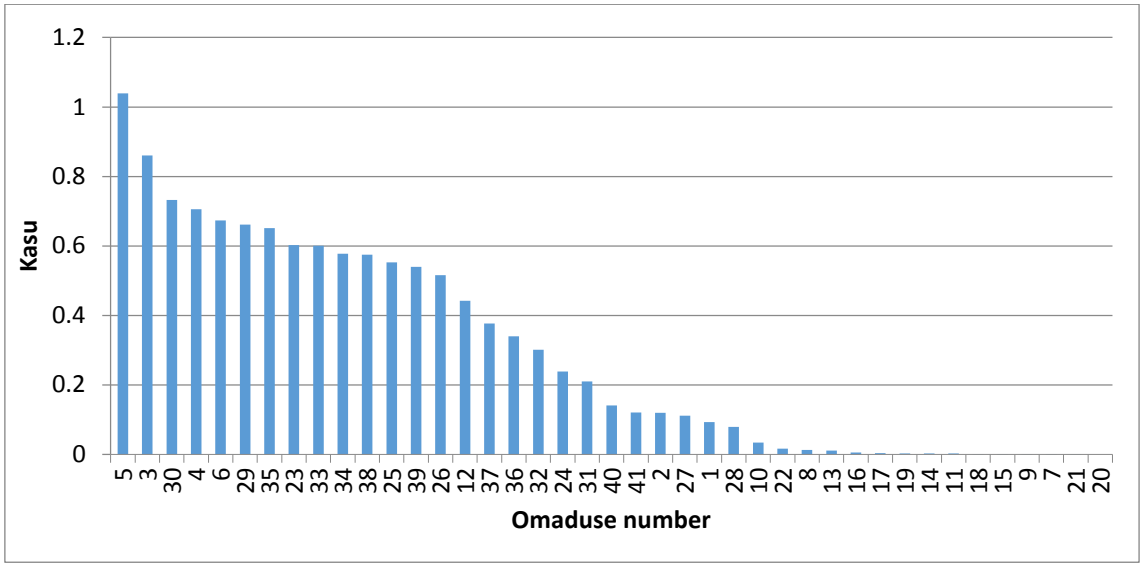
Omadus *O*, millel on erinevad väärtused $(o_1,o_2,\ldots,o_v)$ jagab andmekogumi *v* alamhulkadeks $(S_1,S_2,\ldots,S_v)$, kus S_j on kirjete alamhulk, mille omaduse *O* väärtus on o_j. Lisaks sisaldagu S_j kirjeid S_{ij}, mis kuuluvad klassi *i*. Omaduse *O* entroopia arvutatakse kasutades valemit (11) [14, lk 3].

$$E(O) = -\sum_{j=1}^{v} \frac{S_{1j}+S_{2j}+\cdots+S_{mj}}{S} E(S_{1j}, S_{2j}, \ldots, S_{mj}) \quad (11)$$

Omaduse *O* informatsiooni kasuks on kirjete klassifitseerimiseks vajava informatsiooni entroopia ja omaduse entroopia vahe (12) [14, lk 3].

$$Kasu(O) = E(S_1, S_2, \ldots, S_m) - E(O) \quad (12)$$

Joonisel 2 on kujutatud omaduste kasu kasutades valemit (12), valides uuritavateks klassideks ühenduste 5 põhiklassi. Omadused on esitatud numbriliselt (vt. Tabel 3). Graafikult on näha, et ligi poolte omaduste kasu on suhteliselt minimaalne.



Joonis 2. Omaduste informatsiooni kasu

Tabelis 6 on välja toodud mõlema klasterdamise meetodi tulemused kasutades selleks klastrite arvu 22 ja 13 omadust. Valitud on omadused (5, 6, 12, 23, 25, 28, 29, 30, 33, 34, 35, 38, 39), mille kasu on suurem kui 0,4 (välja arvatud nominaalsed omadused 3 ja 4).

Tabel 6. 13 omadusega klasterdamise tulemused.

Algo-ritm	**CR_{Normal}**	**CR_{DoS}**	**CR_{Probe}**	**CR_{U2R}**	**CR_{R2L}**	**Aeg**	**R**	**CR_{Total}**
MQ	43,14%	95,12%	99,88%	25,00%	56,48%	1,92	60,03	67,44%
LL	35,92%	95,98%	99,90%	28,35%	58,06%	6,93	124,13	63,91%

Võrreldes tabelis 4 ja 5 välja toodud tulemustega on mõlemad algoritmid muutunud ligi 2 korda kiiremaks ning üleüldine täpsus on paranenud. Viimane tuleneb normaalsete ühenduste klasterdamise paranemisest. Kuigi U2R ja R2L rünnakute grupeerimine on muutunud tunduvalt halvemaks, ei mõjuta see suuresti üleüldist täpsust, kuna mõlemate rünnakute sisaldatavus andmekogumis on väga minimaalne.

5.4 Omaduste normaliseerimine

Iga kirje andmekogumis sisaldab 41 omadust, millel on erinevad väärtuse ulatused. Kuna k-keskmine klasterdamine ei tee vahet omaduste väärtuste ulatusest, hakkavad suurema väärtuse ulatusega omadused domineerima klasterdamise protsessis, mis mõjutab algoritmi efektiivsust [1]. Seega on vaja omadused normaliseerida ja panna homogeensesse väärtuste vahemikku. Lisaks numbrilistele omadustele on andmekogumis kasutusel 3 nominaalset omadust. K-keskmiste klasterdamises saab kasutada aga ainult numbrilisi atribuute, järelikult on vaja nominaalsed teisaldada numbrilisteks. Antud töös on normaliseerimiseks kasutatud ja võrreldud kahte meetodit: skaleerimine ja standardiseerimine.

Nominaalsete omaduste numbriliseks muutmisel on antud töös esiteks leitud omaduse samasuguste väärtuste esinemise arv ning pandud need vastavalt esinemisele kahanevasse järjekorda. Nii saame järjekorra, kus kõige enam esinev väärtus on järjekorra ees ja vähem esinev selle lõpus. Nominaalsete omaduste muutmisel numbrilisteks antakse neile eelnevalt kirjeldatud järjekorra number.

Skaleerimine [4] viib atribuutide väärtused kasutaja poolt soovitud vahemikku. Kuna suurem osa omadusi (vt. Tabel 3) on juba vahemikus [0,1], siis on antud töös viidu kõik omaduste väärtused sellesse vahemikku. Meetodi valem (13) on suhteliselt lihtne, kus x on omaduse esialgne väärtus, $min(x)$ on omaduse minimaalne ja $max(x)$ on selle maksimaalne väärtus ning a ja b soovitud vahemik.

$$\acute{x} = a + \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}} \quad (13)$$

Standardiseerimine [16] aitab tsentraliseerida omadused, muutes neid omavahel paremini võrreldavaks. Standardiseerimiseks kasutatakse valemit (14), kus x on omaduse esialgne väärtus, μ on omaduse keskväärtus (15) ja σ omaduse standardhälve (16).

$$\acute{x} = \frac{x - \mu}{\sigma} \quad (14)$$

$$\mu = \frac{1}{n}\sum_{i}^{n} x_i \quad (15)$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(x_i - \mu)^2} \quad (16)$$

Kuna suurem osa eelnevalt valitud 13 omadusest on juba vahemikus [0,1], siis on skaleerimiselt valitud just see vahemik. Mõlema meetodi kasutamisel saadud tulemused on välja toodud tabelis 7 ja 8, kus lisaks eelnevalt leitud 13 tõhusamale omadusele on kasutatud ka 3 nominaalset omadust. Tabelites on standardiseerimiseks kasutatud lühendit Zscore ja skaleerimisel MinMax. Saadud tulemused on märgatavalt erinevad eelnevalt saadud tulemustega. Mõlemad klasterdamise kiirused on järjekordselt muutunud kiiremaks. Lisaks ajale on ka üleüldine täpsus ligi 30% parem, mis tuleneb normaalsete ühenduste klasterdamise täpsuse paranemisest ligi 50% võrra. Halvemaks muutus jällegi U2R ja R2L klasterdamise täpsus. Kasutatud meetoditest on üleüldise täpsuse suhtes miinimum maksimum meetod parem, aga rünnakute klasterdamise täpsus ning aeg on märgatavalt halvem võrreldes standardiseerimisega. Sellest tulenevalt on antud töö eesmärkide saavutamiseks standardiseerimine parem valik, kui lihtsalt soovitud vahemikku viimine.

Tabel 7. MacQueen klasterdamise meetodi tulemused omaduste normaliseerimisel.

Meetod	**CR_{Normal}**	**CR_{DoS}**	**CR_{Probe}**	**CR_{U2R}**	**CR_{R2L}**	**Aeg**	**R**	**CR_{Total}**
Zscore	91,77%	96,65%	86,75%	1,92%	6,51%	0,58	12,22	92,37%
MinMax	97,93%	93,48%	71,24%	3,17%	2,51%	1,49	34,14	93,04%

Tabel 8. Lloyd klasterdamise meetodi tulemused omaduste normaliseerimisel.

Meetod	**CR_{Normal}**	**CR_{DoS}**	**CR_{Probe}**	**CR_{U2R}**	**CR_{R2L}**	**Aeg**	**R**	**CR_{Total}**
Zscore	91,95%	96,48%	85,84%	1,92%	5,96%	2,11	27,36	92,32%
MinMax	97,97%	93,50%	68,08%	3,38%	2,58%	4,92	72,93	92,78%

5.5 Esialgsete klastrite valimine

Lisaks klastrite arvule tagab k-keskmiste klasterdamine erinevaid tulemusi esialgsete klastrite valimisel. Kuna vaikimisi valitakse neid suvaliselt uuritavast andmekogumist tagab klasterdamine väga erinevaid tulemusi. Paremate esialgsete kalstrite valimiseks on anutud töös kasutatud David Arthuri ja Sergei Vassilvitskii'i välja pakutud k-keskmiste++ klasterdamise hoolikat esialgsete klastrite valimise meetodit [3].

K-keskmiste++ algoritm näeb välja järgnevalt [3]:

1. Valitakse klaster C_1 suvaliselt uuritavast andmekogumist.
2. Valitakse uus klaster C_i kasutades selleks tõenäosust $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$ kus $D(x)^2$ on andme lähim eukleidiline kaugus juba valitud klastritest.
3. Sammu 2 korratakse kuni on valitud soovitud arv klastreid.

Tabelis 9 välja toodud tulemustest näha, et hoolikalt valitud esialgsete klastrite korral on tunduvalt paremaks muutunud klasterdamisel saadud halvimad tulemused. Samuti on jällegi vähenenud algoritmi iteratsioonide arv ning sellest tulenevalt klasterdamiseks kulunud aeg. Töös kasutatud hoolikalt valitud esialgsete klastrite meetod on Java koodina välja toodud lisas 3.

Tabel 9. Suvaliselt ja hoolikalt valitud esialgsete klastrite klasterdamise tulemused.

Meetod	**Algo-ritm**	**Halvim CR_{Total}**	**Keskmine CR_{Total}**	**Parim CR_{Total}**	**VPM**	**VNM**	**Aeg**	**R**
Suvaline	MQ	87,52%	92,37%	93,29%	8,23%	6,93%	0,58	12,22
Hoolikalt	MQ	92,00%	92,64%	94,90%	7,63%	7,03%	0,39	5,60
Suvaline	LL	87,50%	92,32%	94,13%	8,05%	7,26%	2,11	27,36
Hoolikalt	LL	92,02%	92,69%	94,79%	7,54%	7,05%	1,06	13,28

5.6 Test andmekogumi klasterdamine

Selles alampeatükis on tabelis 10 ja 11 testimise andmekogumi klasterdamise tulemused, kasutades selleks eelnevates alampeatükkides välja toodud erinevaid optimeerimise meetodeid. Kuna testimise andmekogum on mõeldud treening

andmekogumi põhjal arendatud meetodi hindamiseks, kasutades selleks nn. võõrast andmekogumit, on meetodite kasutamisel jäetud klastrite arv 22-ks ning kasutatud samasuguseid parimaid omadusi, mis treening andmekogumi klasterdamisel.

Nagu treening andmekogumi klasterdamisel on iga optimeerimise meetodi rakendamine muutnud klasterdamise tulemusi ning aega tunduvalt paremaks. Märgatavalt erinevad tulemused võrreldes treening andmekogumi klasterdamisega on U2R ja R2L rünnakute grupeerimise täpsused. U2R ja R2L rünnakute grupeerimise tulemuste paranemises mängib kindlasti suurt rolli nende rünnakute suurem sisaldatavus andmekogumis, kus treening andmekogumis moodustasid nad 0,04% ja 0,79% ning testimise andmekogumis 0,30% ja 12,80% tervest andmekogumis olevatest kirjetest.

Tabel 10. MacQueen klasterdamise tulemused.

Meetod	**CR_{Normal}**	**CR_{DoS}**	**CR_{Probe}**	**CR_{U2R}**	**CR_{R2L}**	**Aeg**	**R**	**CR_{Total}**
Algne	61,28%	98,45%	98,83%	21,87%	81,53%	1,45	81,22	80,09%
13 O	67,40%	99,80%	98,76%	20,90%	81,18%	0,63	100,78	83,11%
Zscore	85,60%	94,97%	94,05%	76,22%	84,96%	0,11	12,77	89,50%
Hoolikalt	85,23%	94,67%	97,91%	88,96%	76,77%	0,08	6,95	88,65%

Tabel 11. Lloyd klasterdamise tulemused.

Meetod	**CR_{Normal}**	**CR_{DoS}**	**CR_{Probe}**	**CR_{U2R}**	**CR_{R2L}**	**Aeg**	**R**	**CR_{Total}**
Algne	56,06%	99,76%	98,86%	23,25%	81,21%	4,06	211,60	78,24%
13 O	55,01%	99,77%	99,08%	25,22%	81,28%	1,47	188,02	77,83%
Zscore	83,68%	88,92%	97,85%	80,66%	61,42%	0,53	53,82	84,08%
Hoolikalt	85,00%	95,54%	96,04%	94,39%	83,17%	0,17	15,25	89,47%

6 Kokkuvõte

Antud töö eesmärgiks oli analüüsida sissetungi avastamise andmekogumi klasterdamisel saadud tulemusi, kasutades selleks k-keskmiste klasterdamist, ning leida võimalikke optimeerimise võimalusi parema tulemuse saavutamiseks.

Välja toodud optimeerimise meetodite rakendamisel selgus, et antud andmekogumi klasterdamise efektiivsuse tõstmiseks on parem kasutada tähtsaimaid ühendusi kirjeldavaid omadusi ning samas normaliseerida nende väärtuste ulatusi. Lisaks nendele tagab hoolikalt valitud esialgsete klastrite meetodi kasutamine paremaid üleüldiseid grupeerimise täpsuseid ja vähendab klasterdamiseks kuluvat iteratsioonide arvu ning sellest tulenevalt grupeerimise kiirust. Välja toodud optimeerimise meetodite rakendamise kombinatsiooni tulemusena muutus treening andmekogumi üleüldine klasterdamise täpsus Lloyd klasterdamise meetodit kasutades 55,52%-lt 92,69%-ni ja MacQueeni klasterdamist kasutades 62,09%-lt 92,64%-ni. Lisaks täpsusele muutus ka mõlema meetodi klasterdamiseks kulunud aeg ligi 10 korda kiiremaks. Treening ja testimise andmekogumi klasterdamise tulemustest selgus, et k-keskmiste klasterdamisel mängib suurt rolli erinevate klasterdavate objektide sisaldatavus andmekogumis.

Antud töös saavutati püstitatud eesmärgid muuta k-keskmiste klasterdamine täpsemaks ning kiiremaks. Töö autor on veendunud, et välja toodud optimeerimise meetoditest on kindlasti kasu tulevastes k-keskmiste klasterdamise töödes.

Antud töös kasutatud meetodite realisatsiooni kood on saadaval https://github.com/StenUD/kmeans.

Kasutatud kirjandus

[1] Clustering high-dimensional data. [WWW] https://en.wikipedia.org/wiki/Clustering_high-dimensional_data (06.12.2016)

[2] DARPA Intrusion Detection Evaluation. [WWW] https://www.ll.mit.edu/ideval/data/1998data.html (27.11.2016)

[3] D. Arthus ja S. Vassilvitskii. k-means++: The Advantages of Careful Seeding. – *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, 1027–1035. [WWW] http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf (10.12.2016)

[4] Feature scaling. [WWW] https://en.wikipedia.org/wiki/Feature_scaling (06.12.2016)

[5] How to add and subtrack values from an average? [WWW] https://math.stackexchange.com/questions/22348/how-to-add-and-subtract-values-from-an-average (08.12.2016)

[6] James B. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. – *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, 1, 281–297. [WWW] https://projecteuclid.org/euclid.bsmsp/1200512992 (08.12.2016)

[7] KDD Cup 1999 Data. [WWW] https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (27.11.2016)

[8] K-means clustering. [WWW] https://en.wikipedia.org/wiki/K-means_clustering (27.11.2016)

[9] L. Morissette ja S. Chartier. The k-means clustering technique: General considerations and implementation in Mathematica. – *Tutorials in Quantitative Methods for Psychology*, 2013, 9(1), 15–24. [WWW] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.159&rep=rep1&type=pdf (08.12.2016)

[10] M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani. A Detailed Analysis of the KDD CUP 99 Data Set. – *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, 1, 1–6. [WWW] http://www.ee.ryerson.ca/~bagheri/papers/cisda.pdf (27.11.2016)

[11] NSL-KDD Data Set. [WWW] https://web.archive.org/web/20150205070216/http://nsl.cs.unb.ca/NSL-KDD/ (18.12.2016)

[12] P. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Chapter 8, Cluster Analysis: Basic Concepts and Algorithms. [WWW] https://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf (27.11.2016)

[13] Sensitivity and specificity. [WWW] https://en.wikipedia.org/wiki/Sensitivity_and_specificity (28.11.2016)

[14] S. Sethuramalingam ja E.R. Naganathan. Hybrid Feature Selection for Network Intrusion. – *International Journal of Emerging Technology and Advanced Engineering*, 2012, 3(5), 1773-1780. [WWW] http://www.enggjournals.com/ijcse/doc/IJCSE11-03-05-035.pdf (06.12.2016)

[15] Stuart P. Lloyd. Least Squares Quantization in PCM. – *IEEE Transaction of Information Theory*, 1982, 28(2), 129–137. [WWW] http://www.cs.nyu.edu/~roweis/csc2515-2006/readings/lloyd57.pdf (27.11.2016)

[16] Standard score. [WWW] https://en.wikipedia.org/wiki/Standard_score (06.12.2016)

Lisa 1 – Lloyd klasterdamine

```
public static void clustering(ArrayList<Data3> dataList, ArrayList<Data3>
centroids) {
final double big = Double.MAX_VALUE;
	double minimum = big;
	double distance = 0.0;
	int cluster = 0;
	boolean moving = true;
	long start = System.currentTimeMillis();
	//k6ik andmed esialgsetesse klastritesse
	for (int i = 0; i < dataList.size(); i++) {
		Data3 data = dataList.get(i);
		minimum = big;
		//l2himasse klastrisse
		for (int j = 0; j < numOfClusters; j++) {
			distance = Distance.distAll(data, centroids.get(j));
			if (distance < minimum) {
				minimum = distance;
				cluster = j;
			}
		}
		//m22rab klastri
		data.setCluster(cluster);
	}
	//arvutab k6ikide klastrite uued keskpunktid
	CalcCentroids.lloydCentroidsAll(dataList, centroids, numOfClusters);
	int r = 0;
	//andmete liigutamine klastrite vahel seni kuni ykski anne enam
klastreid ei vaheta
	//v6i on saavutatud maksimaalne iteratsioonide arv
	while (moving && r < iterations) {
		moving = false;
		for (int i = 0; i < dataList.size(); i++) {
			Data3 tmp = dataList.get(i);
			minimum = big;
			//leiab l2hima klastri
			for (int j = 0; j < numOfClusters; j++) {
				distance = Distance.distAll(tmp, centroids.get(j));
				if (distance < minimum) {
					minimum = distance;
					cluster = j;
				}
			}
			//kui l2him klaster on esialgsest erinev
```

```
                    if (tmp.getCluster() != cluster) {
                        //m22rab uue klastri
                        tmp.setCluster(cluster);
                        moving = true;
                    }
                }
                //arvutab k6ikide klastrite uued keskpunktid
                CalcCentroids.lloydCentroidsAll(dataList, centroids,
numOfClusters);
                r++;
        }
return;
}
```

Lisa 2 – MacQueen klasterdamine

```
public static void clustering(ArrayList<Data3> dataList, ArrayList<Data3>
centroids) {
final double big = Double.MAX_VALUE;
	double minimum = big;
	double distance = 0.0;
	int cluster = 0;
	boolean moving = true;
	long start = System.currentTimeMillis();
	//K6ik andmed esialgsetesse klastritesse
	for (int i = 0; i < dataList.size(); i++) {
		Data3 data = dataList.get(i);
		minimum = big;
		//l2himasse klastrisse
		for (int j = 0; j < numOfClusters; j++) {
			distance = Distance.distAll(data, centroids.get(j));
			if (distance < minimum) {
				minimum = distance;
				cluster = j;
			}
		}
		//M22rab klastri
		data.setCluster(cluster);
		//Arvutab klastri uue keskpunkti
		CalcCentroids.mqAddAll(data, cluster, centroids, clustersList);
	}
	int r = 0;
	int tmpCluster = 0;
	// Andmete liigutamine klastrite vahel seni kuni ykski anne enam
klastreid ei vaheta
	// v6i on saavutatud maksimaalne iteratsioonide arv
	while (moving && r < iterations) {
		moving = false;
		for (int i = 0; i < dataList.size(); i++) {
			Data3 tmp = dataList.get(i);
			minimum = big;
			//leiab l2hima klastri
			for (int j = 0; j < numOfClusters; j++) {
				distance = Distance.distAll(tmp, centroids.get(j));
				if (distance < minimum) {
					minimum = distance;
					cluster = j;
				}
			}
```

```
                        //kui l2him klaster on esialgsest erinev
                        if (tmp.getCluster() != cluster) {
                                tmpCluster = tmp.getCluster();
                                //arvutab esialgse klastri uue keskpunkti
                                CalcCentroids.mqSubAll(tmp, tmpCluster, centroids,
clustersList);
                                tmp.setCluster(cluster);
                                //arvutab uue klastri uue keskpunkti
                                CalcCentroids.mqAddAll(tmp, cluster, centroids,
clustersList);
                                moving = true;
                        }
                }
                r++;
        }
        return;
}
```

Lisa 3 – K-keskmiste++ esialgsete klastrite valimine

```
public static ArrayList<Data3> initialize(ArrayList<Data3> theList) {
	long start = System.currentTimeMillis();
	Random random = new Random();
	ArrayList<Integer> intList = new ArrayList<>();
	ArrayList<Double> prob = new ArrayList<>();
	ArrayList<PlusPlus> plusList = new ArrayList<PlusPlus>();

	int max = theList.size();
	float r = 0;
	double probability = 0.0;
	int size = theList.size();
	int centroid = 0;
	double distance = 0.0;
	int location = 0;
	double distanceTotal = 0.0;

	int rCent = random.nextInt(max);
	intList.add(rCent);
	centTmp.add(theList.get(rCent));

	for (int i = 0; i < numOfClusters; i++) {
		DatasInClusters d = new DatasInClusters();
		clustersList.add(d);
	}

	for (int i = 0; i < size; i++) {
		PlusPlus p = new PlusPlus();
		plusList.add(p);
	}

	for (int x = 0; x < numOfClusters - 1; x++) {
		//leiab andmele l2hima klastri juba olemasolevatest klastritest
		for (int i = 0; i < size; i++) {
			distance = Distance.dist16(theList.get(i),
centTmp.get(centroid));
			//kui see klaster on l2hemal paneb l2himaks kauguseks
			if (distance < plusList.get(i).getDistance()) {
				plusList.get(i).setDistance(distance);
			}
			distanceTotal += plusList.get(i).getDistance();
		}
		//arvutab andmete t6en2osust esialgse klastri valimiseks
		for (int i = 0; i < size; i++) {
```

```
                    probability = plusList.get(i).getDistance() /
distanceTotal;
                    prob.add(probability);
             }
             //suvaline arv 0st 1ni
             r = random.nextFloat();
             //esialgse klastri m22ramine
             for (int i = 0; i < size; i++) {
                    r -= prob.get(i);
                    //kui suvaliseks arvuks on j22nud 0 v6i v2iksem
                    //ja valitud anne ei ole juba klastriks
                    //m22ratud, valib selle andme klastriks
                    if (r <= 0 && !intList.contains(i)) {
                           location = i;
                           break;
                    }
             }
             distanceTotal = 0;
             prob = new ArrayList<Double>();
             centTmp.add(theList.get(location));
             intList.add(location);
             centroid += 1;
      }

      ArrayList<Data3> re = new ArrayList<>();
      re = Transform.copy(centTmp);
      return re;
}
```