LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

KVANTU MAŠĪNMĀCĪŠANĀS – KLASTERĒŠANAS ALGORITMA IMPLEMENTĒŠANA

BAKALAURA DARBS

Autore: **Elizabete Koturova**

Studenta apliecības Nr.: ek16064

Darba vadītājs: Dr.dat. Maksims Dimitrijevs

RIGA 2020

ANOTĀCIJA

Klasterēšana ir populārākā bez uzraudzības mašīnmācīšanās metode, kuras pamatprincips ir noteikt datu punktu sadalījumu apakšgrupās vai kopās, lai vienā un tā pašā klastera novērojumiem būtu zināma līdzība. Kvantu mašīnmācīšanās šobrīd ir aktuālā tēma, un kvantu algoritmi tiek izmantoti lai paātrinātu un/vai izlabotu klasiskus algoritmus.

Darba ietvaros tiek analizēts un implementēts kvantu mašīnmācīšanās algoritms klasterēšanai, tas tiek salīdzināts ar klasisko algoritma implementāciju. Algoritms tiek implementēts, izmantojot Python programmēšanas valodu un Qiskit bibliotēku kvantu skaitļošanai, kas ļauj kvantu skaitļošanas kodu laist gan uz simulatoriem, gan uz IBM kvantu ierīcēm, kas ir pieejamas mākonī. Rezultātos tiek secināts, ka kvantu klasterēsanas algoritms neuzlabo klasiska algoritma rezultātus, tomēr paātrina tā darbību.

Atslēgvārdi: Kvanti, mašīnmācīšanās, Qiskit, klasterēšana, Python

ABSTRACT

QUANTUM MACHINE LEARNING – CLUSTERING ALGORITHM IMPLEMENTATION

Clustering is the most popular unsupervised machine learning method, which deals with data point distribution into subsets or clusters so that observations in the same cluster have observed similarity. Quantum machine learning is a relevant sphere and quantum algorithms are used to speed up and/or optimize classic algorithms.

In this study, a quantum machine learning algorithm for clustering is analyzed, implemented, and compared to the classic algorithm implementation. Algorithm is implemented using Python programming language and Qiskit library for quantum computing, that allows running quantum code both on simulators and real IBM quantum device which run in the cloud. The conclusion is that clustering algorithm doesn't improve the results of classic implementation, but it speeds up the calculation.

Keywords: quantum, machine learning, Qiskit, clustering, Python

SATURS

VĀRDNĪCA

Jupyter Notebook – atvērtā koda lietojumprogramma, kas ļauj izveidot un kopīgot dokumentus, kas satur kodu, vienādojumus, vizualizācijas un tekstu.

Python – augsta līmeņa vispārējas nozīmes programmēšanas valoda, kas ir vērsta uz izstrādātāju produktivitātes un koda lasāmības uzlabošanu.

Qiskit – kvantu skaitļošanas atvērtā koda platforma, kas nodrošina rīkus kvantu programmu izveidošanai un pārvaldīšanai, kā arī to darbināšanai kvantu ierīču prototipos IBM Q Experience vai lokālos simulatoros.

IBM QE – mākoņa platforma, kas lietotājam dod piekļuvi pie IBM kvantu procesoru prototipiem.

Hiperparametrs – parametrs, kura vērtība tiek iestatīta pirms mācību procesa sākuma.

QRAM (angliski *Quantum Random Access Memory*) – kvantu brīvpiekļuves atmiņa.

SSE (angliski *Sum of Squared Error*) – kvadrātisko kļūdu summa.

ID – identifikācijas numurs.

IEVADS

Mašīnmācīšanās kļūst arvien aktuālākā cilvēku dzīvē, plašā pielietojumu klāstā, sākot ar sastrēgumu analīzi līdz pašbraucošām automašīnām. Palielinās uzdevumu skaits, kas tiek risināti ar mašīnmacīšanos palīdzību [1].

Zinot, ka ir paredzams, ka mūsu sabiedrībā ģenerēto datu apjoms pieaugs, ir nepieciešami jaudīgāki informācijas apstrādes veidi. Kvantu skaitļošana ir daudzsološa jauna paradigma, lai veiktu aprēķināšanu ātrāk. Pēdējos gados ir bijuši priekšlikumi kvantu mašīnu apgūšanas algoritmiem, kas var piedāvāt ievērojamu (eksponenciālu vai lielu polinomiālo) paātrinājumu, salīdzinot ar attiecīgiem klasiskiem algoritmiem [2].

Darba ietvaros tiek analizēta bez uzraudzības mācīšanās problēma – klasterēšana, kas no dotas datu kopas, kura ir padota kā N vektori, piešķir vektorus k grupām, tādejādi, lai vienā klasterā būtu vienādi vektori. Bieži, lai atrastu izmērīto vektoru līdzību ir izmantots Eiklīda attālums, bet arī citas metrikas varētu būt izmantotas, tas ir atkarīgs no izskatāmas problēmas.

Darba ietvaros tiek implementēts un aprakstīts q-vidējais – kvantu algoritms klasterēšanai, kas ir kvantu alternatīva klasiskajam k-vidējam. Abu algoritmu darbība ir savstarpēji salīdzināta, lai noteiktu katra algoritma efektivitāti.

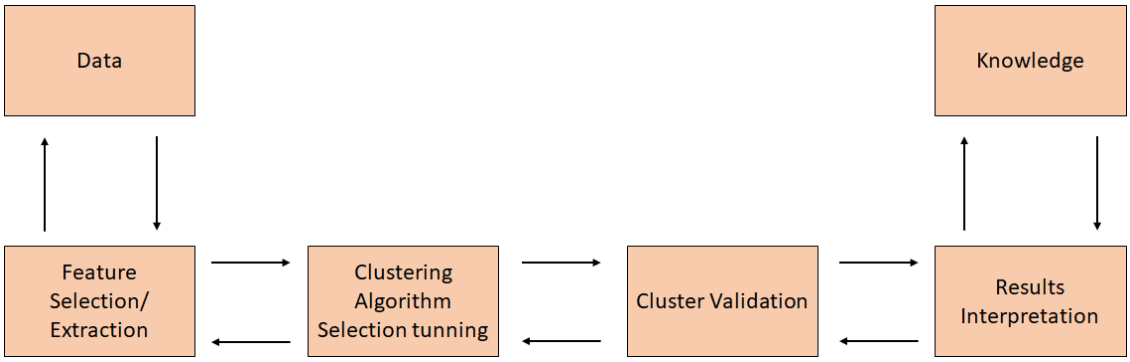
Dokuments ir strukturēts un sastāv no trīs daļām: pirmajā daļā ir aprakstīts mašīnmacīšanas algoritms klasterēšanai, otrajā daļā ievads kvantu skaitļošana un kvantu klasterēšanas algoritms, un trešajā daļā ir aplūkots algoritmu klasiskā un kvantu implementēšana, ka arī šie algoritmi tiek salīdzināti.

1. MAŠĪNMĀCĪŠANĀS

Lielāko daļu no mašīnmācīšanās uzdevumiem var sadalīt divās daļās – uzraudzīta mācīšanās (angliski *supervised learning*) un bez uzraudzības mācīšanās (angliski unsupervised learning). Mācoties ar uzraudzību, ir dati ar marķējumiem, balstoties uz kuriem ir nepieciešams kaut ko paredzēt (prognozēt), un dažas hipotēzes. Mācoties bez uzraudzības ir tikai dati bez marķējumiem, kuriem ir nepieciešams atrast līdzību [1].

1.1. Ievads

Bez uzraudzības mācīšanās ir mašīnmācīšanās tehnika, kur nav pieejama informācija par datu marķējumiem, to vietā ir nepieciešams atļaut modelim strādāt patstāvīgi, lai izdarītu secinājumus par datiem. Modelis strādā ar neiezīmētajiem datiem, lai mācīšanās algoritms pats varētu atrast to struktūru.



1.1. att. **Bez uzraudzības mašīnmācīšanās modelis**

Attēlā 1.1. ir redzams vispārīgs klasterēšanas modelis, kas raksturo to darbību. Sākumā ir tikai dati, tad notiek pazīmes izvēle un sāk darbību klasterēšanas algoritms, tad ir klasteru validācija, rezultātu vizualizācija un pēdējais elements modelī – izgūtie secinājumi.

Bez uzraudzības mācīšanās galvenie uzdevumi ir:

- Datu kopu segmentēšana pēc dažiem atribūtiem;
- Anomāliju atrašana (novērojumi, kas nepieder nevienai grupai);
- Datu kopu vienkāršošana, apkopojot mainīgos ar līdzīgiem atribūtiem.

Kopsavilkumā, galvenais mērķis ir izpētīt datu raksturīgo (un parasti slēpto) struktūru.

Šie uzdevumi var būt izteikti ar divām problēmām, kurus bez uzraudzības mācīšanās mēģina atrisināt. Šīs problēmas ir:

- Klasterēšana;
- Dimensiju skaita samazināšana.

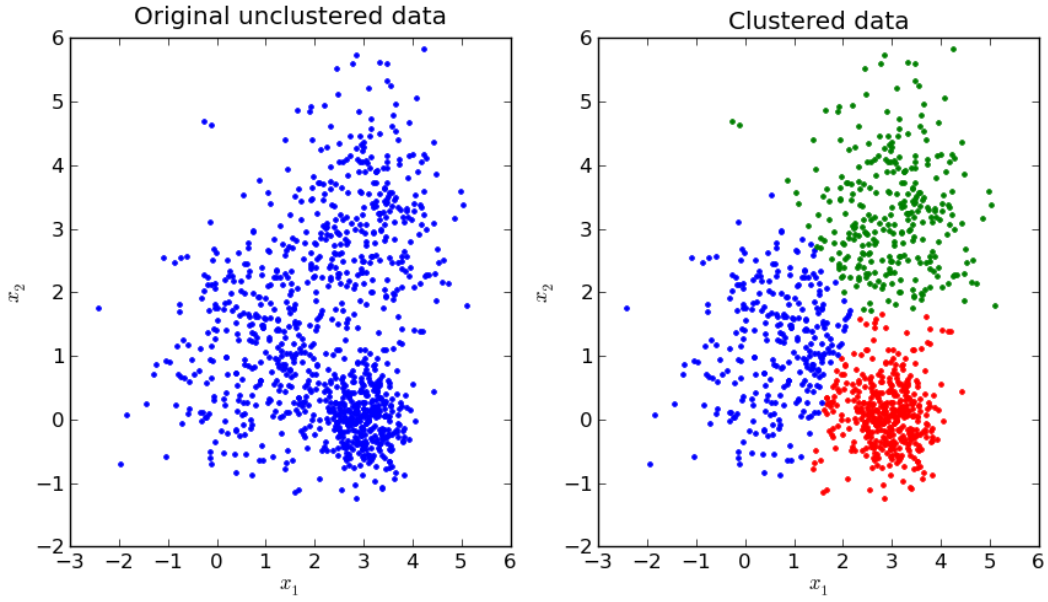
Darba ietvaros tiek aplūkota klasterēšanas problēma.

1.2. Klasterēšana

Klasterēšana ir bez uzraudzības mācīšanās vispopulārākā metode, kur dati tiek grupēti balstoties uz datu punktu līdzību, tās grupas sauc par klasteriem. Klasterēšanai ir daudz pielietojumu reālā dzīvē, kurus var izmantot dažādās situācijās.

Datu klasterēšana ir uzdevums, kas sadala grupas uzdotus datus (objektus) tā, lai katrs klasteris sastāvētu no līdzīgiem objektiem, un dažādu klasteru objekti būtiski atšķirtos viens no otra. Klaterēšanas galvenais uzdevums ir, izmantojot visus pieejamos datus, paredzēt objektu atbilstību kādai no klasēm, tādejādi veidojot klasterus.

Tiek aplūkots piemērs ar datiem, kas tiek sadalīti trīs klasteros (sk. 1.2. at..). Attēla kreisajā pusē ir dati pirms klasterēšanas un labajā pusē – pēc klasterēšanas.



1.2. att. **Klasterēšanas piemērs**

Bez uzraudzības mācīšanās algoritmiem ir ļoti plašs lietojums un tie ir ļoti noderīgi, lai atrisinātu dažāda veida problēmas, piemēram, anomāliju noteikšanas, dokumentu grupēšanas, vai arī lai atrastu klientus ar kopīgām interesēm, pamatojoties uz to pirkumiem.

1.3. Klasterēšanas kritērijs

Lai salīdzinātu divus objektus, ir nepieciešams kritērijs, uz kura pamata notiks salīdzinājums. Parasti šis kritērijs ir attālums starp objektiem.

Ir daudz attāluma mērvienības, dažas no tām ir:

- Eiklīda attālums – visizplatītākais attālums, kas ir ģeometriskais attālums daudzdimensionālajā telpā;
- Eiklīda attāluma kvadrāts – palīdz atrast precīzāko attālumu starp vairāk attālinātiem objektiem;
- Manhetenas attālums – ir koordinātu vidējo starpība, visbiežāk šis attālums rada tādus pašus rezultātus, kā parastais Eiklida attālums;
- Čebiševa attālums - šis attālums var būt noderīgs, ja ir nepieciešams definēt divus objektus kā "atšķirīgus", ja tie atšķiras vienā koordinātā (= vienā dimensijā).
- Pakāpeniskais attālums –izmantots, lai palielinātu vai samazinātu ļoti atšķirīgu objektu svaru, kuri atrodas dažādās dimensijās.

Attāluma izvēle (līdzības kritērijs) ir atkarīga no pētnieku izvēles un problēmas specifikas. Klasterēšanas rezultāti var ievērojami atšķirties, izvēloties dažādas metodes.

Apakšnodaļās tiek aprakstītas visbiežāk izmantoti klasterēšanas paveidi.

1.4. K-vidējais

K-vidējām algoritmam ir viegla implementēšana un efektīvs rezultāts, tie ir galvenie iemesli, kāpēc tas ir tik izplatīts.

K-vidējais algoritms, kā jau bija minēts, tiecas atrast un grupēt klasēs datu punktus, kuri ir līdzīgi. Algoritmā šī līdzība ir saprotama kā pretstats attālumā starp datu punktiem. Jo tuvāk datu punkti ir, jo vairāk līdzīgi tie ir un, visticamāk, tie piederēs pie viena un tā paša klastera.

Klastera centroīds ir klastera viduspunkts. Centroīds ir vektors, kurā ir viens skaitlis katram mainīgajam, kur katrs skaitlis ir mainīgā vidējais lielums šajā klasterī. Centroīdu var uzskatīt par klastera daudzdimensionālo vidējo vērtību.

Lai aprēķinātu attālumu starp datu punktiem, ir jāizmanto 1.3. nodaļā aprakstīto distances aprēķinu metodi.

K-vidējā algoritmā visbiežāk ir lietota kvadrātiskā Eiklīda attāluma metode. Formula, ar kuru ir aprēķināms šī veida attālums diviem datu punktiem x un y m-dimensiju telpā, ir aplūkota zemāk:

$$d(x,y)^2 = \sum_{j=1}^{m} \left(x_j - y_j\right)^2 = \|x - y\|_2^2$$

Noradītā formulā ar j ir apzīmēta j-tā dimensija no piemērā dotajiem punktiem x un y.

Klasteru inerce ir kvadrāta kļūdas summa (SSE). Katram datu punktam kļūda ir attālums līdz tuvākajam klasterim un tiek attēlota šādi [1]:

$$SSE = \sum_{\mathrm{i}=1}^{\mathrm{K}} \sum_{x \epsilon C_i} dist^2\,(m_i, x),$$

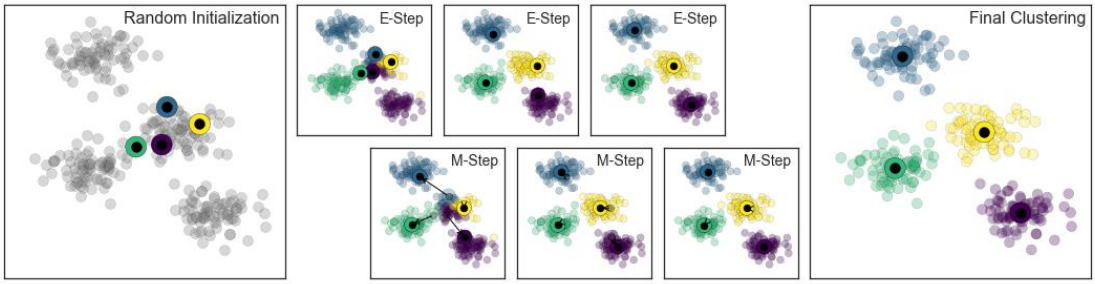
kur x ir datu punkts klasterā C_i, un m_i ir raksturīgais punkts klasterā C_i.

K-vidēju algoritmu var interpretēt kā algoritmu, kas mēģina samazināt klastera inerces koeficientu [1].

Lai apstrādātu pētāmos datus, k-vidējais algoritms sāk ar pirmo izvelēto grupu no nejauši izvelētiem centroīdiem, kuri ir izmantoti kā sākumpunkti katram klasterim, un pēc tam veic iteratīvos (atkārtojošos) aprēķinus, lai optimizētu centroīdu pozīcijas.

Algoritma soļi (sk. 1.3. att):

1. Ir nepieciešams izvelēties/atrast klasteru skaitu – *k*;
2. Algoritms nejauši izvēlās centroīdu katram klasterim;
3. Katrs datu punkts būs piešķirts pie tuvākā centroīda (izmantojot Eiklida attālumu);
4. Tiek aprēķināta klastera inerce;
5. Jauns centroids tiek aprēķināts kā vidējais no punktiem, kas pieder centroīdam, jeb izrēķina datu punktu kvadrātisko kļūdas minimumu katram klastera centram, pārvietojot centru šī punkta virzienā;
6. Algoritms atgriežas pie 3. soļa.



1.3. att. **K-vidēja algoritma piemērs**

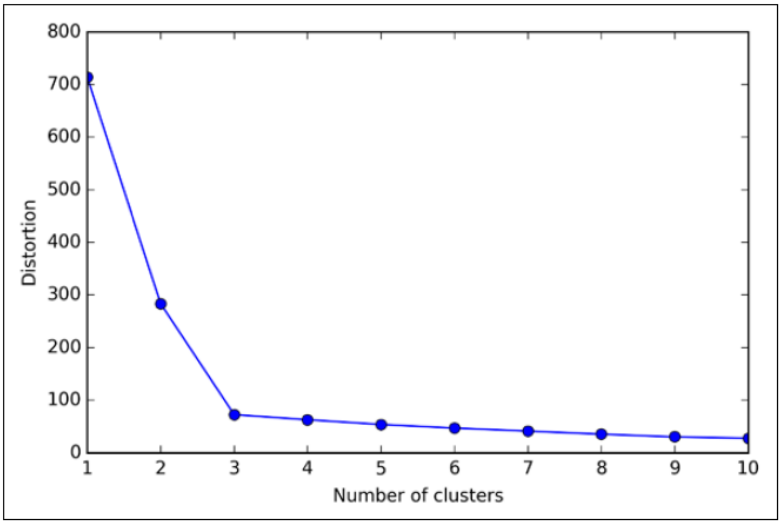
Attēlā 1.3. ir apskatāma centroīdu atrašanas vietas mainīšana. No sākuma centroīdu vieta tiek nejauši izvelēta, tad dati tiek piešķirti klasteram un centroīdu atrašana vieta katrā algoritma iterācijā tiek mainīta, līdz momentam kad centroīdu vieta vairs nemainīsies un algoritma darbība nebeigsies.

K-vidējā hiperparametri:

- Klasteru skaits – klasteru un centroīdu skaits, kurš ir jāģenerē;
- Iterāciju maksimālais skaits;
- Sākumu skaits – cik reižu algoritms tiks palaists ar dažādiem centroīdiem.

Pareizā klasteru skaitu izvēle ir viena no svarīgākiem soļiem k-vidējā algoritmā. Eksistē dažādas metodes, lai atrastu šo skaitu, tomēr visplašākā ir elkoņu metode.

Elkoņa metodes ideja ir palaist k-vidējo algoritmu ar izmantotu datu kopu, k vērtības diapazonā, un katrai k vērtībai aprēķināt kvadrātisko kļūdu summu.



1.4. att. **Elkoņa metode**

Tad var aplūkot līniju diagrammu ar kvadrātisko kļūdu katrai k vērtībai. Jā līniju diagramma izskatās kā roka, tad "elkonis" šajā rokā ir labākā k vērtība. Ideja ir tāda, ka ir nepieciešams mazais

SSE, bet tam ir dilstoša tendence uz 0, kad k palielinās (SSE ir 0, kad k vērtība ir vienāda ar kopas datu punktu skaitu, tāpēc ka katram datu punktam ir savs klasteris un neeksistē kļūda starp tā punkta un klastera centroīdu). Mērķis ir izvelēties zemo k vērtību, kuram joprojām ir zems SSE, un elkonis parasti parāda, kur sakas samazināšana, palielinot k vērtību [6]. (sk. 1.4. att.).

Iepriekšminētajā piemērā, pēc elkoņa metodes k būs vienāds ar 3. K-vidējais algoritms ir vairāk noderīgs, kad iepriekš ir zināms klasteru skaits.

2. KVANTU DATORIKA UN MAŠĪNMĀCĪŠANĀS

Nodaļā ir aprakstīti kvantu skaitļošanas pamati un aplūkoti ar kvantu mašīnmācīšanos saistītie algoritmi. Lai apgūtu kvantu skaitļošanas modeli, ir nepieciešams iepazīties ar galveniem kvantu jēdzieniem.

2.1. Ievads

Kvantu mašīnmācīšanās ir topošā starpdisciplinārā pētniecības joma, kurā krustojas kvantu fizika un mašīnmācīšanās.

Kvantu mašīnmācīšanās (QML) ir kvantu informācijas apstrādes pētījumu apakšdisciplīna, kuras mērķis ir attīstīt kvantu algoritmus, kas mācās no datiem, lai uzlabotu esošās mašīnmācīšanās metodes. Kvantu algoritms ir procedūra, kuru var realizēt kvantu datorā – ierīcē, kas izmanto kvantu teorijas likumus, lai apstrādātu informāciju.

2.2. Ievads kvantu skaitļošanā (kvantu datorika)

Visas skaitļošanas sistēmas pamatojas uz fundamentālu spēju uzglabāt un manipulēt ar informāciju. Mūsdienas datori manipulē ar atsevišķiem bitiem, kas glabā informāciju binārā stāvokli, jeb 0 vai 1. Kvantu datori izmanto kvantu mehāniskās parādības, lai manipulētu ar informāciju. Lai to izdarītu ir nepieciešami kvantu biti jeb kubiti.

2.2.1. Kubits un stāvoklis

Kvantu datorikā kubits ir binārā bita analogs. Kubits ir kvantu daļiņas, kuriem ir piemītoša īpašība: izņemot standartos 0 un 1, kubits var atrasties starp 0 un 1, ko sauc par superpozīciju – divu kubitu bāzes lineāra kombinācija.

Par kvantu stāvokli saucas jebkāds iespējamais stāvoklis, kurā var atrasties kvantu sistēma. Kvantu stāvoklis ar tam atbilstošo amplitūdu tiek pierakstīts šādi:

$$\alpha|s\rangle$$

kur α – kompleksais koeficients, un s – kvantu stāvokļa nosaukums. Pēdējais viena kubita gadījumā sastāv no viena simbola, piemēram: «0», «1». Tad kvantu stāvoklī ir tādi objekti kā $|0\rangle$ un $|1\rangle$. Stāvokļus var attēlot ar vektoriem.

Tad ir iespējams aplūkot kubita vektoru skatu, diviem bāzes kubitiem $|0\rangle$ un $|1\rangle$:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

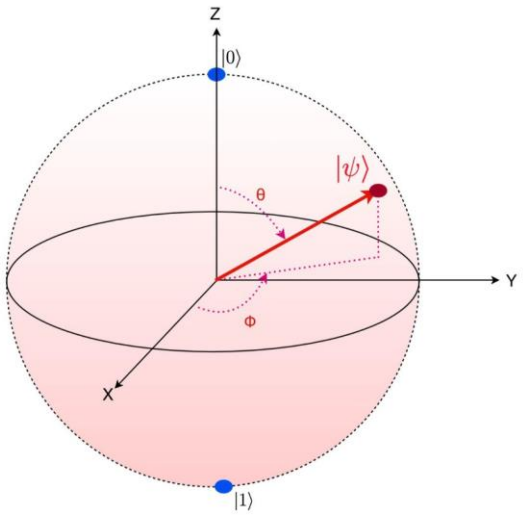
$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Kubits līdz mērījumiem var atrasties vienlaicīgi abos stāvokļos, tādu parādību sauc par superpozīciju, tāpēc to ir ierasts apzīmēt ar sekojošo izteiksmi:

$$\alpha|0\rangle + \beta|1\rangle$$

kur α un β — kompleksie skaitļi, kas apmierina nosacījumu $|\alpha|^2 + |\beta|^2 = 1$. Šos skaitļus sauc par varbūtību amplitūdām. Tas raksturo, cik lielā mērā kubits ir attiecīgajā pamatstāvoklī [3].

To visu ir vieglāk saprast, ja paskatīties uz grafisko skatu. Neskatoties uz to, ka līdz šim punktam, lai ilustrētu kvantu stāvokļu koncepcijas un kubitu izmanto dažus simbolus un manipulācijas ar tiem, to var attēlot grafiski. Šajā gadījumā katrs kubits ir vienības vektors, un viņa attēlojums bāzē ir šī vektora projekcija uz nejauši izvelētas ortogonālās asēs. Kubitu grafiskā attēlojumā ir arī apskatāma θ (teta), kas ir 0 un 1 amplitūdu attiecība. Teta ir pilnība saistīta ar varbūtību iegūt 0 vai 1 pēc mērījuma (sk. 2.1. att.)



2.1. att. **Kubitu grafiskais attēlojums, reālo skaitļu koeficientu gadījumā**

Skatoties uz attēlu 2.1. ir skaidrs kāpēc $\alpha^2 + \beta^2 = 1$ — ir nosacījums ka riņķa rādiuss ir vienāds ar vieninieku – tā ir Pitagora teorēma.

2.2.2. Kvantu operācija (Geits)

Kvantu skaitļošanas modelī katra aprēķināšana notiek ar geitu palīdzību. Kvantu skaitļošanas terminos geits ir "unitāra operācija". No sākuma qiskit bibliotēkā visi kubiti sākas no

stāvokļa 0, tad kubiti tiek pārveidoti ar kvantu ķēdem, kas veic aprēķināšanas, un sastāv no vairākiem operācijām jeb geitiem.

Viens no geitiem, kas ir bieži lietots kvantu algoritmos, ir Adamāra operācija. Adamāra operāciju apzīmē ar H, un tai ir sekojoša matrica:

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

Adamāra transformācija izskatās sekojoši:

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$|1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Nākamā aplūkota transformācija ir mijmaiņas (angliski *swap*), kas attēlojas tādā matricā un transformācijās:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} |00\rangle \rightarrow |00\rangle \\ |01\rangle \rightarrow |10\rangle \\ |10\rangle \rightarrow |01\rangle \\ |11\rangle \rightarrow |11\rangle \end{matrix}$$

Transformācija CNOT realizē XOR līdzīgu operāciju:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} |00\rangle \rightarrow |00\rangle \\ |01\rangle \rightarrow |10\rangle \\ |10\rangle \rightarrow |11\rangle \\ |11\rangle \rightarrow |10\rangle \end{matrix}$$

Pēdēja darba ietvaros aplūkota transformācija ir U – patvaļīga viena kubita transformācija, kas ir izmantota SWAP transformācijā un tiek attēlota sekojoši:

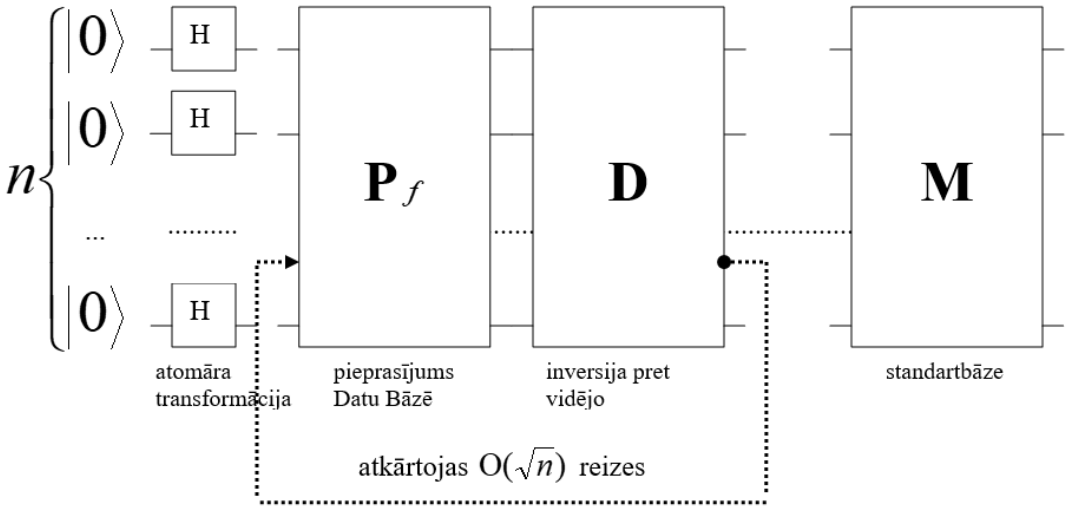
$$\begin{bmatrix} cos\left(\frac{\theta}{2}\right) & -e^{i\lambda} sin\left(\frac{\theta}{2}\right) \\ e^{i\phi} sin\left(\frac{\theta}{2}\right) & e^{i\lambda + i\phi} cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

Šī matrica ir viena kubita transformācijas vispārīgā forma [16].

2.3. Grovera algoritms

Grovera algoritms atrod elementu nesakārtotā kopā kvadrātiski ātrāk, nekā teorētiski to ir iespējams atrast ar klasisko algoritmu [15]. Meklējamais elements definē funkciju: funkcija tiek novērtēta kā patiesa uz elementa, ja tas ir meklētāis elements. Grovera algoritms izmanto orākula iekšējo izsaukumu, kas nosaka šīs funkcijas vērtību, vai funkcija ir patiesa konkrētai instancei. Mērķis ir izmantojot minimālo izsaukumu skaitu, atrast visus elementus ar vērtību = 1 (patieso).

Zemāk var apskatīt Grovera algoritma shēmu (sk. 2.2. att.) [3]:



2.2. att. **Grovera algoritma shēma**

Algoritmu var sadalīt sekojošos soļos:

1. Likt visus kubitus superpozicijā izmantojot Adamāra transformāciju;
2. Grovera algoritma iterāciju pielietojums: šī iterācija sastāv no diviem transformācijas secīgiem izsaukumiem – orākula un difūzijas, kas tiek detalizētāk aplūkoti zemāk; šī iterācija izpildās $\sqrt{2^n}$ reizes;
3. Mērījums – pēc nepieciešamo reižu Grovera iterāciju pielietošanas, ir nepieciešams izmērīt kubitu reģistru ieeju. Ar lielu varbūtību mērāmā vērtība izdod norādījumu uz meklējamo parametru. Ja ir nepieciešams palielināt atbildes uzticamību, algoritms tiek izsaukts vēl dažas reizes un tiek aprēķinātā pareizas atbildes kopīga varbūtība.

Grovera algoritma orākulam ir nepieciešams invertēt to kvantu stāvokļa fāzi, kurai atbilst meklējama vērtība x. Tas nozīme, ka orākulam izejā ir jāizdod vērtība $(-1)^{f(x)}|x\rangle$. Fāžu koeficients $(-1)^{f(x)}$ ir tieši tāds, jo $f(x)$ = 1 tikai tad, kad funkcija ieejā saņem meklējamo vērtību x, un tikai tādā gadījumā fāžu koeficients kļūst vienāds ar -1. Citādi sakot, orākuls U_w darbojas sekojoši [4]:

$$U_w|w\rangle = -|w\rangle$$
$$U_w|x\rangle = |x\rangle, x \neq w$$

Savukārt, difūzijas operāciju ķēde ir trīs operāciju savienojums, no kuras divas ir daudzkubitu Adamāra transformācija. Starp tām ir speciālā operācija, kas īsteno kubitu inversiju pret vidējo vērtību. Transformāciju matricā visi elementi diagonālē ir $-1 + \frac{2}{N}$, bet pārējie ir $\frac{2}{N}$ [3]:

$$D = \begin{pmatrix} -1+\frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & -1+\frac{2}{N} & \cdots & \frac{2}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{2}{N} & \cdots & \cdots & -1+\frac{2}{N} \end{pmatrix}$$

2.4. Kvantu k-vidējais (q-vidējais)

Vairums no kvantu klasterēšanas algoritmiem ir balstīti uz Grovera meklēšanas algoritmu [6]. Šie klasterēšanas algoritmi nodrošina paātrinājumu, salīdzinot ar klasiskiem algoritmiem, bet neuzlabo klasterēšanas procesa rezultātu. Tas ir pamatots uz pieņēmuma, ka ja optimālo risinājumu klasterēšanas problēmai atrašana ir NP – sarežģīta, tad kvantu datori arī nevarēs to atrisināt tieši polinomiālajā laikā. Jā izmanto QRAM, tad ir iespējams eksponenciālais paātrinājums [5].

Kvantu k-vidēja versija ļauj ātri aprēķināt attālumu, kā arī dod eksponenciālo paātrinājumu klasiskam algoritmam, ja ieejas un izejas vektori ir kvantu stāvokļi. Paātrinājumu nodrošina fakts, ka pirmkārt, klasiskie dati ir N-dimensiju vektora (N- dimensiju skaits) formā, tie var būt ielādēti kvantu stāvokļos vairāk nekā log2 N kubitos, bet kad dati ir saglabāti kvantu brīvpiekļuves atmiņā (QRAM), tad ielādēšana aizņem O(log2 N) soļus. Kad dati ir kvantu formā, tie var būt apstrādāti ar dažādiem kvantu algoritmiem, kas aizņem O(poly(log N)) laiku. Kvantu datorā attāluma aprēķināšana starp N-dimensiju vektoriem aizņem O(log N) laiku [11].

Kvantu k-vidēja algoritma implementēšana ir viena no kvantu klasterēšanas algoritmiem, kurā nav obligāti jāizmanto Grovera algoritmu. Tomēr vienkāršākais kvantu k-vidēja variants, kur tiek aprēķināti centroīdi un vektori tiek piešķirti pie tuvāka centroīda, arī kā klasiskajā versijā, bet izmantojot Grovera meklēšanu, lai atrastu tuvāko centroīdu [6].

Kvantu un klasiska k-vidēja galvenā atšķirība ir redzama darba praktiskajā daļā.

3. PRAKTISKĀ DAĻA

Darba ietvaros tiek izmantots k-vidējais klasterēšanas algoritms un implementēts q-vidējais (kvantu k-vidējais) algoritms. Nodaļā ir aprakstīts katrs algoritms, to implementēšana un īpatnības, kā arī tiek veikta algoritmu analīze un salīdzinājums.

Nodaļā ir aplūkots:

1. q-vidēja realizēšana Fišera varavīksnes datiem;
2. Kvantu algoritma realizēšanas ietekme uz k-vidēja algoritma gala rezultātu;
3. Pārbaudīt, kāds darba laiks būs šiem algoritmiem.

3.1. Fišera varavīksnes

K-vidēja algoritma uzdevums ir sadalīt datus pa klasteriem, šim uzdevumam tiek paņemta klasiskā datu kopa – Fišera varavīksnes. Tā ir datu kopa klasifikācijas uzdevumam, uz kura piemēra Ronalds Fišers 1936.gadā parādīja savas diskriminējošās analīzes metodes darbu.

Datu kopa sastāv no datiem par 150 varavīksnes ziediem, kas iekļauj sevī trīs sugas, 50 eksemplārus no katras šķirnes. Tas ir:

- *Iris setosa*
- *Iris virginica*
- *Iris versicolor*

3.1. att. ***Iris setosa***

3.2. att. ***Iris virginica***

3.3. att. ***Iris versicolor***

Datu kopa ietver sevī četras raksturlīknes un kolonnu ar pareizo varavīksnes sugu. Ziedu raksturīgi parametri ir:

- Kauslapas garums (cm);
- Kauslapas platums (cm);
- Vainaglapas garums (cm);

- Vainaglapas platums (cm).

Datu kopas pirmie seši ieraksti ir aplūkoti zemāk paradītajā tabulā:

3.1. tabula

Fišera varavīksnes datu kopa

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa

3.2. K – vidējais algoritms

K-vidēja algoritma implementēšana tiek veikta Jupyter Notebook izstrādes vidē Python programmēšanas valodā.

K-vidēja algoritma koda pamats tika paņemts no mācības avota [7], un tika papildināts ar elkoņu metodes implementēšanu.

3.2.1. K-vidējais divdimensiju telpā

Pēc datu nolasīšanas, mainīgajam X, kas ir K-means bibliotēkas iebūvētais parametrs, piešķir divas kolonnas no datu kopas, kas attiecīgi būs x un y asis, grafikā, kur tiek attēloti datu punkti un aprēķināti klasteri (sk. 3.4. att.)

```
#Select the annual income and the spending score columns
X=dataset.iloc[:, [1,2]].values
print(X)
```

```
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]
 [5.4 3.9]
 [4.6 3.4]
 [5.  3.4]
 [4.4 2.9]
 [4.9 3.1]
 [5.4 3.7]
```

3.4. att. **Datu ielasīšana**

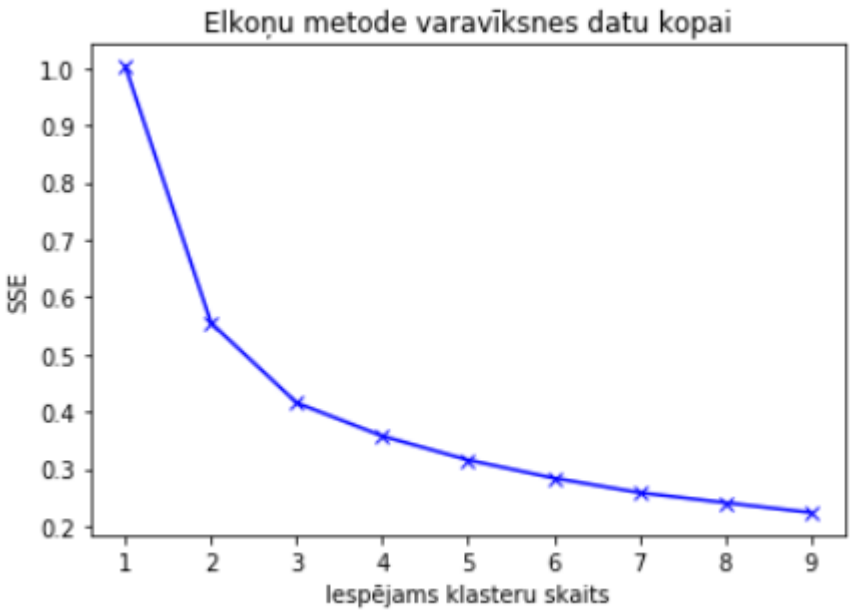
Klasteru skaits ar elkoņa metodes aprēķināšanu notiek ar Python bibliotēku Kmeans. Šī bibliotēka iekļauj sevī palīgmetodes un parametrus, lai atvieglotu algoritma implementēšanu. Kā jau minēts nodaļā 1.3.1. klasteru skaits tiek aprēķināts ar katra klastera kvadrātisko kļūdu aprēķināšanu.

```
In [46]: from sklearn.cluster import KMeans
         wcss =[]
         for i in range (1,11):
             kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter =150, n_init = 10, random_state = 0)
             kmeans.fit(X)
             wcss.append(kmeans.inertia_)
```

```
In [49]: # Plot the graph to visualize the Elbow Method to find the optimal number of cluster
         %matplotlib notebook
         plt.plot(range(1,11),wcss)
         plt.title('Elkoņa metode')
         plt.xlabel('Klasteru skaits')
         plt.ylabel('SSE')
         plt.show()
```

3.5. att. **Elkoņa metode varavīksnes kopai**

Attēlā 3.5. redzams koda fragments, kur masīvā "wcss" (Kvadrātisko kļūdu summa katram punktam) tiek pievienotas vērtības klasteru skaitam no 1 līdz 11. Otrajā koda daļā tiek izvadīta līniju diagramma, lai noteikt optimālo klasteru skaitu (sk. 3.6. att,).



3.6. att. **Elkoņu metode varavīksnes datu kopai**

Pēc grafika apskatīšanas ir redzams, ka optimālais klasteru skaits ir 3, kas sakrīt ar reālo pareizo klasteru skaitu, jo datu kopā ir trīs ziedu sugas, kuras veido trīs klasterus.

Kad klasteru skaits ir noteikts, var uzsākt datu sadalīšanu klasteros (sk. 3.7. att.)

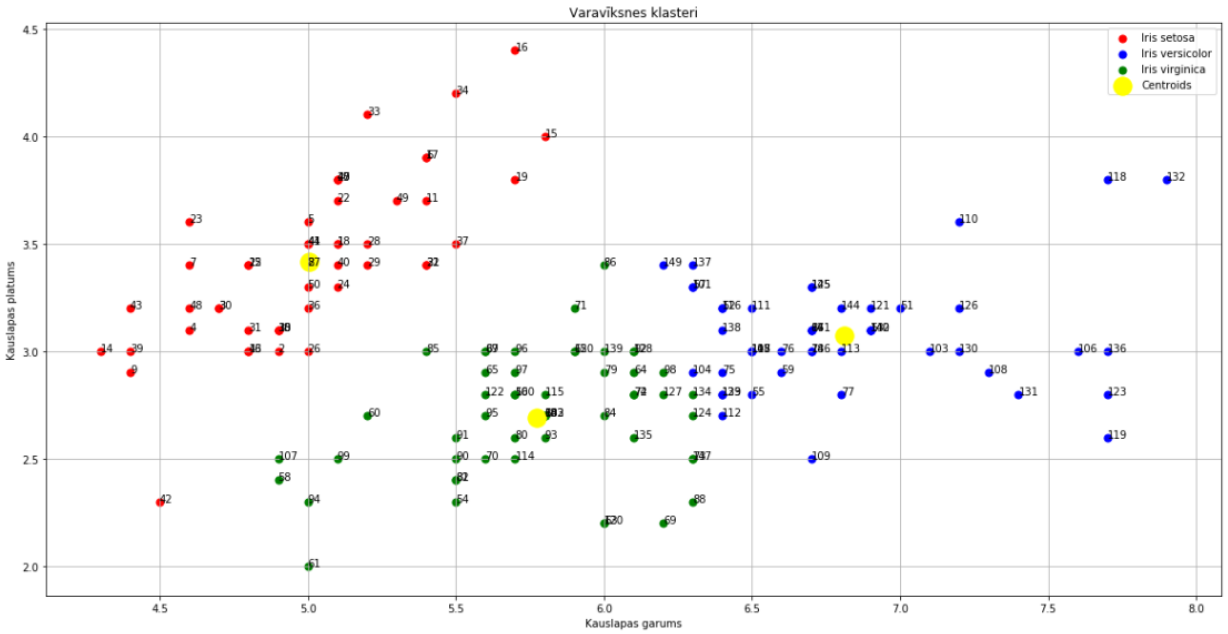
```
In [38]: #5 According to the Elbow graph we deterrmine the clusters number as #3. Applying k-means algorithm to the X dataset.
kmeans = KMeans(n_clusters=3, init ='k-means++', max_iter=300, n_init=10,random_state=0 )

y_kmeans = kmeans.fit_predict(X)
```

3.7. att. **Datu sadalīšana klasteros**

Metode *fit_predict()* paredz datu punktu piederēšanu noteiktam klasterim, tas notiek maksimāli 300 reizes, līdz momentam, kad centroīdu atrāšanas vieta ar katru iterāciju netiek ievērojami mainīta.

Pēc datu sadalīšanas klasteros, tiek izvadīts grafiks ar datu punktiem, kur katram datu punktam tiek pierakstīts "ID", lai būtu vieglāk noskaidrot, vai punkts tiešām pieder šim klasterim (sk. 3.8. att.).



3.8. att. **Klasiskā k-vidēja pielietojums varavīksnes datu kopai**

Rezultātā katram no trīm klasteriem jāpieder attiecīgiem ziediem ar ID:

- Pirmais klasteris: 1-50
- Otrais klasteris: 51-100
- Trešais klasteris: 101-150

Attēlā 3.8 var redzēt, ka ne visi punkti ir piešķirti pareizajām klasterim, tas arī ir apskatāms rezultātu matricā (sk. 3.9. att.), tomēr tas var mainīties atkarīgi no datu kopas izmantotajām kolonnām. Attēlā ir paņemtas kolonnas ar nosaukumiem "Kauslapas garums" un "Kauslapas platums".

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
 1 1 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
```

3.9. att. **Rezultātu matrica k-vidējam algoritmam divdimensiju telpā**

Kā redzams attēlā 3.9., klasteru numuri katram datu punktam ir sajaukti un nav attēloti augošā secība, tāpēc var secināt, ka ne visi datu punkti tiek piešķirti pareizajam klasterim.

3.2.2. K-vidējais trīsdimensiju telpā

Klasteru sadalījums ir atkarīgs arī no dimensiju skaita, tāpēc šī nodaļā ir paradīts klasteru sadalījums trīsdimensiju telpā. Kā analizējami dati tiek paņemti trīs kolonnās:

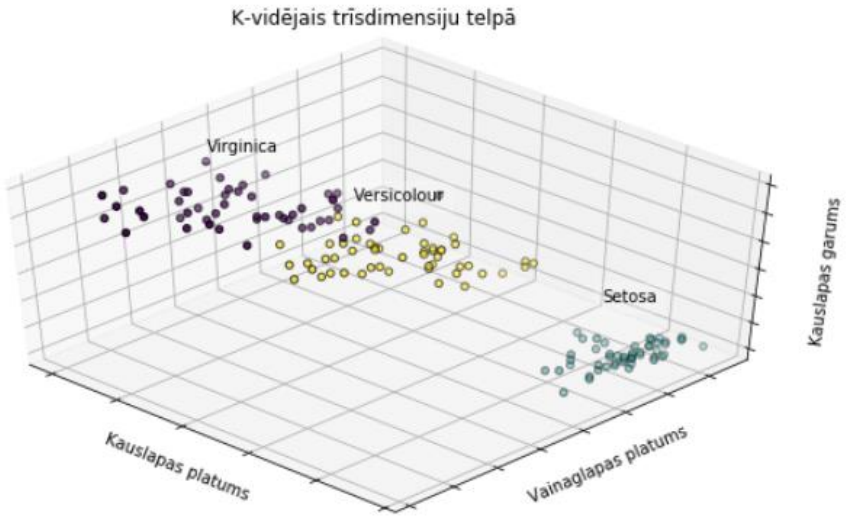
- “Kauslapas garums”;
- “Kauslapas platums”;
- “Vainaglapas platums”.

Atkārtojot iepriekšēja algoritma soļus, bet analizējot trīs kolonnas, rezultātā tiek izvadīta matrica ar datu punktu klasteru numuriem (Skatiet attēlu 3.10):

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
```

3.10. att. **Rezultātu matrica trīs dimensiju telpā**

Šajā matricā klasteru numuri netiek sajaukti, bet nav augoša secība, jo algoritms tiek veikts trīs dimensiju telpā, tomēr katram klasterim pieder tieši 50 ziedu sugas, tas nozīme, ka visi datu punkti tiek sadalīti pareizi.



3.11. att. **K-vidējais trīsdimensiju telpā**

Attēlā 3.11 ir apskatāma k-vidēja vizualizācija varavīksnes datu kopai trīsdimensiju telpā.

3.3. Q-vidēja algoritma implementācija

Ir dažādas metodes, kā implementēt kvantu k-vidējo algoritmu. Darba ietvaros tiek implementēts variants ar kvantu attāluma aprēķināšanu.

Pirmkods ir implementēts un glabāts, izmantojot "IBM Quantum Experience" tīmekļa pakalpojumus [9]. Šī platforma ir mākoņa lietojumprogramma, lai programmētu uz reālās kvantu aparatūras vai augstas veiktspējas simulatoros. Programmēšana notiek "Qiskit Notebook", kas ir uz Jypiter Notebook balstītā izstrādes vide, kas dod iespēju rakstīt kodu ar Qiskit bibliotēkas palīdzību.

Koda sakumā ir nepieciešams importēt qiskit bibliotēkas funkcionālu, un pieslēgties IBM kontam (sk. 3.12. att.).

```
In [7]:
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

3.12. att. **Qiskit bibliotēkas importēšana un pieslēgšanā IBM kontam**

Nākamais, kas ir nepieciešams, ir aprēķināt mainīgo θ (teta) vērtību, kas ir nepieciešama U_3 operācijai, kura ir aprakstīta 2.2.2. nodaļā. Šai transformācijai ir nepieciešami trīs parametri:

$$u3(\theta, \phi, \lambda) = U(\theta, \phi, \lambda)[11],$$

kur parametri ϕ un λ kodā ir vienādi ar π, un θ ir leņķis, kas tiek aprēķināts ar formulu:

$$\theta = \arctan\left(\frac{y}{x}\right), 0 < \theta < 1$$

Formulas parametri x un y ir analizējamas datu kolonnās, attiecīgi kā klasiskajā k-vidējā tas ir "Kauslapas garums" un "Kauslapas platums". Vērtība, kas piešķirta mainīgajam θ, tiek saglabāta un pievienota datu kopas tabulas kolonnā ar nosaukumu "Teta" (Skatiet attēlu 3.13).

```
In [8]:
df_data_1 = pd.read_csv('Iris.csv')
df_data_1.head()
data=df_data_1

datat['Teta']=np.arctan(dataset['SepalWidthCm'].values/dataset['SepalLengthCm'].values)

data.head(20)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Teta
0	1	5.1	3.5	1.4	0.2	Iris-setosa	0.601455
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0.549374
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0.597758
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0.593003
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0.624023
5	6	5.4	3.9	1.7	0.4	Iris-setosa	0.625485

3.13. att. **Pievienotā kolonna Teta un tās aprēķināšana**

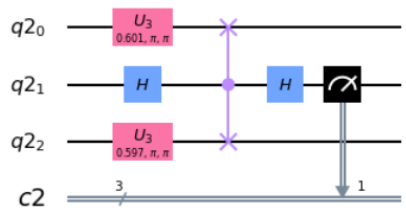
Pēc šīm darbībām nepieciešami dati ir nolasīti un sākās galvenā algoritma daļa, kur tiek aprēķināts attālums un centroīdi.

3.3.1. Attāluma aprēķināšana (swapTest operation)

Attāluma aprēķināšana q-vidējā algoritmā tiek nodrošināta ar diviem svarīgiem punktiem:

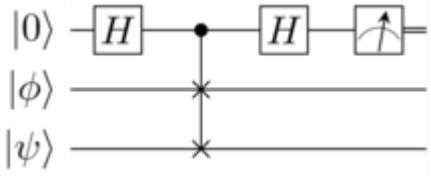
- SwapTest operāciju ķēde
- Attāluma aprēķināšana

Operāciju ķēde SwapTest ir procedūra, kas tiek lietota, lai pārbaudītu divu kvantu stāvokļu starpību [12]. SwapTest sastāv no Adamāra operācijas, Controlled Swap operācijas un atkārtotas Adamāra operācijas. Kodā tas ir pielietots pēc U_3 operācijas (2.2.2.), kura tiek pielietota diviem kvantu stāvokļiem, kuri pēc tam tiek salīdzināti. Tā operāciju ķēde ir izdrukāta ar komandu *circuit_drawer(qc)*, kur *qc* ir kvantu ķēde (quantum circuit), kas iekļauj sevī kvantu reģistru uz trīs kubitem [12][13] (sk. 3.14. att.).



3.14. att. **Kvantu ķēde q-vidēja kodā**

SwapTest procedūra izskatās sekojoši (sk. 3.15. att.):



3.15. att. **SwapTest kvantu ķēde**

Šai operāciju ķēdei ir sekojoši soļi:

- Inicializācija:

Tiek inicializēti divi stāvokļi $|a\rangle$ un $|b\rangle$, kā arī kontroles kubits $|0\rangle$, rezultātā ir stāvoklis:

$$|\psi_0\rangle = |0, a, b\rangle$$

- Šim stāvoklim tiek pielietota Hadamāra operācija, rezultāta ir superpozīcija:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0, a, b\rangle + |1, a, b\rangle)$$

- Tiek pielietota SWAP operācija $|a\rangle$ un $|b\rangle$ stāvoklim, kas samaina a un b, un paredz ka kontroles kubits ir stāvoklī $|1\rangle$:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0,a,b\rangle + |1,b,a\rangle)$$

- Kontroles kubitam tiek pielietota otra Hadamāra operācija:

$$|\psi_3\rangle = \frac{1}{2}|0\rangle(|a,b\rangle + |b,a\rangle) + \frac{1}{2}|1\rangle(|a,b\rangle - |b,a\rangle)$$

- Mērījums

Tiek mērīts kontroles kubits. Varbūtība, ka mērījams kontroles kubits ir stāvoklī $|0\rangle$ ir:

$$P(|0\rangle) = \left|\frac{1}{2}\langle 0|0\rangle(|a,b\rangle + |b,a\rangle) + \frac{1}{2}\langle 0|1\rangle(|a,b\rangle - |b,a\rangle)\right|^2$$

$$= \frac{1}{4}|(|a,b\rangle + |b,a\rangle)|^2$$

$$= \frac{1}{4}(\langle b|b\rangle\langle a|a\rangle + \langle b|a\rangle\langle a|b\rangle + \langle a|b\rangle\langle b|a\rangle + \langle a|a\rangle\langle b|b\rangle)$$

$$= \frac{1}{2} + \frac{1}{2}|\langle a|b\rangle|^2$$

Tādējādi galīgā stāvoklī ir veiksmīgi saistīti $\langle a|b\rangle$ ar kontroles kubitu mērāmo varbūtību. Varbūtība $P(|0\rangle)$ = 0.5 nozīme ka $|a\rangle$ un $|b\rangle$ stāvokļi ir ortogonāli, kamēr varbūtība $P(|0\rangle)$ = 1 nozīmē, ka stāvokļi ir identiski. Operāciju ķēdi jāatkārto vairākas reizes, lai iegūtu pareizāko varbūtību.

Visas izdarītas manipulācijas ir apskatāmas zemāk paradītajā koda fragmentā (sk. 3.16. att.):

```
In [16]:
def getQuantumDistance(theta_1,theta_2, shots):

        pi = math.pi
        qreg = QuantumRegister(3)
        creg = ClassicalRegister(3)
        qc = QuantumCircuit ( qr, cr, name="k_means")

        qc.h(qr[1])

        qc.u3(theta_1, pi, pi, qreg[0])
        qc.u3(theta_2, pi, pi, qreg[2])
        qc.cswap(qr[1], qreg[0], qreg[2])
        qc.h(qreg[1])
        qc.measure(qreg[1], creg[1])

        backend = Aer.get_backend('qasm_simulator')
        job_exp = execute(qc, backend=backend, shots=shots)
        result = job_exp.result()
```

3.16. att. **SwapTest koda fragments**

Attēla 3.17. ir redzams kvantu stāvoklis (ar amplitūdam), kas sanāk pirms mērjījuma, un tas tiek izmantots attāluma aprēķināšanai. Lai to dabūtu tika izmantots simulātors "statevector_simulator". Galvenokārt otrais kubits stāvoklī 0 ir ar lielāko varbūtību, tāpēc kā teta1 un teta 2 parametri ir gandrīz vienādi, kas atbilst cerībai.

```
[ 0.91285976+0.j -0.28203877+0.j  0.        +0.j -0.00106951+0.j
 -0.28203877+0.j  0.08713795+0.j  0.00106951+0.j  0.        +0.j]
| 0 0 0 > ampitude: 0.9128597643292058 , probability: 0.8333129493311732
| 0 0 1 > ampitude: -0.28203877480335227 , probability: 0.07954587049257605
| 0 1 0 > ampitude: 0.0 , probability: 0.0
| 0 1 1 > ampitude: -0.001069511376320248 , probability: 1.143854584078431e-06
| 1 0 0 > ampitude: -0.28203877480335227 , probability: 0.07954587049257605
| 1 0 1 > ampitude: 0.08713794795900945 , probability: 0.007593021974507039
| 1 1 0 > ampitude: 0.001069511376320248 , probability: 1.143854584078431e-06
| 1 1 1 > ampitude: 0.0 , probability: 0.0
```

3.17. att. **SwapTest operāciju rezultāts**

Nākamajos trīs soļos ir aprakstīta Eiklīda attāluma aprēķināšana starp diviem vektoriem a un b.

1) Tiek sagatavoti divi stāvokli:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0,a\rangle + |1,b\rangle)$$

$$|\phi\rangle = \frac{1}{\sqrt{Z}}(|a||0\rangle + |b||1\rangle)$$

Kur $Z = |a|^2 + |b|^2$

2) Ar SwapTest operāciju ķēdi palīdzību tiek aprēķinātas $\langle\psi|\phi\rangle$

3) Izmantojot sekojošo vienādojumu, tiek aprēķināts Eiklīda attālums

$$Attālums = 2Z|\langle\psi|\phi\rangle|^2$$

3.3.2. Grovera algoritms (Grover optimization)

Lai atrastu tuvāko centroīdu tiek izsaukta Grovera optimizācija, kas ir balstīta uz Grovera algoritmu [11].

1) Inicializācija: inicializācija notiek tādā pašā veidā kā klasiskajā k-vidēja algoritmā. Tiek izvēlēts klasteru skaits k un nejauši izvelēti k klasteru centroīdi.
2) Pirmais cikls: katrai cikla iterācijai tiek izsaukta attāluma aprēķināšana, lai aprēķinātu attālumu pie katra klastera centroīda, pēc tam tiek izsaukta Grovera optimizacija, kura atrod tuvāko centroīdu.
3) Otrais cikls: ciklā ir j = {1,…,k} iterācijas un tiek aprēķinātas datu punktu vidējais katram klasterim j. Aprēķināta vērtība kļūst par jauno klasteru centroīdu. (Skatiet attēlu 3.18.).

```
for j in range(0,K):
    centroids[j,:]=1/C[j]*centroids[j,:]
```

3.18. att. **Koda fragments ar jauno klasteru centroīdu aprēķināšanu**

Šis algoritms tiek palaists no ārēja cikla un katrs datu punkts tiks piešķirts pie tuvākā centroīda. Kad visi datu punkti tiek piešķirti pie klasteriem, klasteru centroīdi tiek aprēķināti atkārtoti. Vis process tiks atkārtots līdz momentam, kad vairs nebūs izmaiņas centroīdu aprēķināšanās.

3.3.3. Vizualizācija

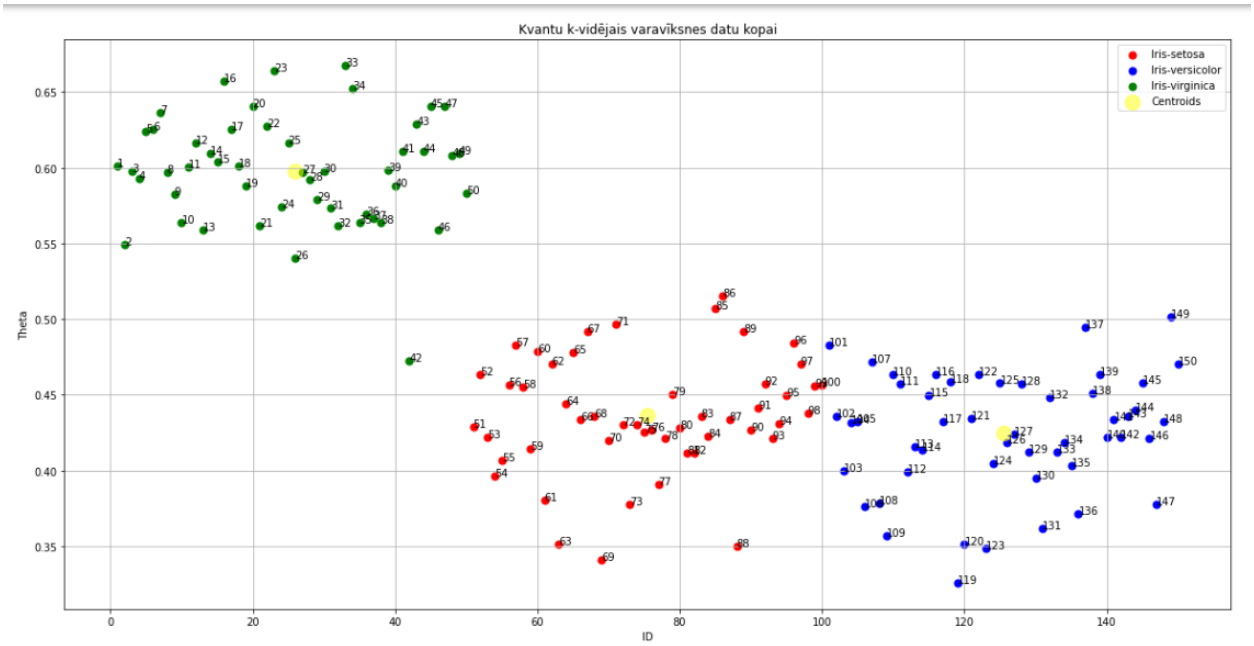
Koda pēdējā daļa ir visvienkāršāka, un ietver sevī datu apstrādāšanu un grafika vizualizāciju.

Pēc visas metodes izsaukšanas un nepieciešama skaita iterācijām, tiek izvadītas centroīdu koordinātes un rezultātu matrica, kur katram datu punktam ir piešķirts klasteru numurs no nulles līdz diviem. Ja rezultātu matricā klasteru vērtības nav sajauktas un tiek izvadītas pēc kārtas, tad klasteru sadalījums ir veiksmīgs un datu punktiem ir piešķirts pareizais klasteris (Skatiet attēlu 3.19).

```
[[ 26.          0.59777426]
 [ 75.5         0.43678636]
 [125.5         0.42500332]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Att. 3.19. **Centroīdu koordinātes un rezultātu matrica**

Kā ir redzams attēlā 3.19. klasterēšanas algoritms ir veiksmīgs un izdod sagaidīto rezultātu. Pēdējais solis algoritma implementēšanā ir klasteru grafiku izvade, tas notiek līdzīgi ka klasiskajam algoritmam.(Skatiet attēlu 3.20.).



Att. 3.20. **Kvantu k-vidējais varavīksnes datu kopai**

Rezultātā ir izvadīts grafiks ar ID vērtībām uz X asi un Teta vērtībām y asi, kas atvieglo grafika analizēšanu, jo visi datu punkti tiek attēloti secīgi.

3.4. Algoritmu analīze (salīdzinājums)

Tabulā 3.2. ir paradīti trīs algoritmu darbības rezultāti:

Tabula 3.2.

Algoritmu rezultāti

Algoritms	Datu sadalīšana klasteros
K-vidējais divdimensiju telpā	Rezultāts ir tuvs pie reāla datu punktu sadalījuma, tomēr otram un trešajam dažas ziedu sugās tika piešķirtas nepareizajam klasterim.
K-vidējais trīsdimensiju telpā	Rezultāts atbilst reālam datu punktu sadalījumam

Q-vidējais	Rezultāts atbilst reālam datu punktu sadalījumam

Analizējot iegūtus rezultātus var secināt, ka pareizais vai nepareizais sadalījums klasteros nav atkarīgs no kvantu vai klasiskas realizēšanas. Tomēr visi rezultāti ir balstīti uz varbūtības, bet kvantu q-vidēja algoritmā varbūtību aprēķināšana var būt palaista vairākas reizes, kas dod palielināto varbūtības pareizību, tas nozīme, ka kvantu algoritma rezultāti var būt nedaudz precīzāk nekā klasiskā algoritma rezultāti.

Klasiskais K-vidēja algoritms ir algoritms, kas aprēķina attālumu starp N-dimensiju vektoriem. Pateicoties efektīvai attāluma aprēķināšanai q-vidējais sasniedz eksponenciālo paātrinājumu. Klasiskā k-vidēja algoritma implementēšanā tiek veikta ar Kmeans bibliotēkas palīdzību, kas ir balstīta uz Loida klasterēšanas algoritma [10][14]. Loida algoritmam laika sarežģītība ir O(MNk) (M - kopējais iterāciju skaits, N – dimensiju vektori, k – klasteru skaits), bet kvantu versijai laika sarežģītība ir O(log(N)Mk), kur papildu priekšrocību nodrošina Grovera algoritma izmantošana. Tas var būt izskaidrots, ņemot vērā, ka N dimensiju klasiskā informācija ir kodēta, izmantojot log2 N kubitus.

REZULTĀTI

Analizējot kvantu un klasisko k-vidēja algoritma darbību ir parādīts, ka kvantu vai klasiska realizācija neietekmē klasterēšanas rezultātus, tie var būt pareizi, vai pietuvināti pareizajam sadalījumam, gan klasiskajā, gan kvantu realizācijā. Tomēr kvantu algoritms var būt nedaudz precīzāks par klasisko k-vidēja algoritmu. Klasterēšanas rezultātus ietekmē vairāki faktori: klasteru izvelētais skaits, datu apjoms, kuri no datu parametriem tiks analizēti, cik dimensiju ir telpā, attāluma aprēķināšanas veids.

Tomēr ir paskaidrots kāpēc kvantu algoritms ir ātrāks, izpildot algoritma darbību O(log(N)Mk) laikā, kamēr klasiskā versija izpilda to pašu algoritmu O(MNk) laikā.

SECINĀJUMI

Darba ietvaros tiek izstrādāta un veiksmīgi palaista uz kvantu simulatora k-vidēja algoritma kvantu versija. Ir veiksmīgi apgūta IBM mākoņa platforma un Qiskit bibliotēka kvantu algoritmu realizēšanai un palaišanai. Tika analizēta, izpētīta un aprakstīta divu algoritmu darbība un to laika sarežģītība, ka arī noteikts, kāpēc kvantu k vidējam ir eksponenciālais paātrinājums.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. V. Roman, "Unsupervised Machine Learning: Clustering Analysis" March 6, 2019 [tiešsaite]. - [Atsauce 20.05.2020.] Pieejams: https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e
2. В.Р. Душкин, *Квантовые вычисления и функциональное программирование*, Москва, 2014, стр. 37-55. [V.R. Duškin, *kvantovie vičislenija i fukcionaljnoje programmirovanije*] (krievu val.)
3. A. Ambains, *Grovera algoritms*: *lekcija 14-02*, Latvijas Universitāte, 2020
4. J. Hui, "QC — Grover's algorithm" Dec 10, 2018 [tiešsaiste]. - [Atsauce 20.05.2020] Peejams: https://medium.com/@jonathan_hui/qc-grovers-algorithm-cd81e61cf248
5. P. Wittek, *Quantum Machine Learning. What Quantum Computing Means to Data Mining*, University of Borås Sweden, 2014, p. 100-101.
6. R. Gove, "Using the elbow method to determine the optimal number of clusters for k-means clustering", Des 26, 2017 [tiešsaiste]. –[Atsauce 20.05.2020]. Pieejams: https://bl.ocks.org/rpgove/0060ff3b656618e9136b
7. S. Girgin, "K-Means Clustering Model in 6 Steps with Python", May 22, 2019 [tiešsaiste]. - [Atsauce 15.05.2020] Pieejams: https://medium.com/pursuitnotes/k-means-clustering-model-in-6-steps-with-python-35b532cfa8ad
8. IBM Quantum Experience, *Qiskit Notebook*, 2020, [tiešsaiste]. – [Atsauce 01.05.2020] Pieejams: https://quantum-computing.ibm.com/
9. S.U. Khan, *Quantum K means Algorithm*, Sweden: Massachusetts Institute of Technology, 2019
10. IBM Quantum Experience, *Advanced single qubit gates*, 2020, [tiešsaiste]. – [Atsauce 20.05.2020] Pieejams: https://quantum-computing.ibm.com/docs/guide/wwwq/advanced-single-qubit-gates
11. M. Schuld, M. Fingerhuth, F. Petruccione, *Implementing a distance-based classifier with a quantum interference circuit*, South Africa: University of KwaZulu-Natal, 2017.
12. Scikit learn documentatiom, *Kmeans*, 2020, [tiešsaiste]. – [Atsauce 20.05.2020]. Pieejams: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

13. Luciano Bello, Jim Challenger, Andrew Cross, Ismael Faro, Jay Gambetta, Juan Gomez, Ali Javadi-Abhari, Paco Martin, Diego Moreda, Jesus Perez, Erick Winston, and Chris Wood, *Qiskit/qiskit-tutorial*. 2018, [tiešsaiste].- Atsauce [01.05.2020]. Pieejams: https://github.com/qiskit/qiskit-tutorial.
14. S. Lloyd, M. Mohseni, P. Rebentrost, *Quantum algorithms for supervised and unsupervised machine learning*, USV: Massachusetts Institute of Technology, 2013.

Bakalaura darbs „Kvantu mašīnmācīšanās – klasterēšanas algoritma implementēšana” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____ Elizabete Koturova

Rekomendēju/nerekomendēju darbu aizstāvēšanai *(nederīgo svītro vadītājs)*

Vadītājs: Zinātniskais asistents Dr.dat. Maksims Dimitrijevs _____ 25.05.2020.

Recenzents: profesors Dr. dat. Juris Vīksna

Darbs iesniegts Datorikas fakultātē 25.05.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____