УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Јаков Петровић

Београд 2020.

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА
ИНФОРМАЦИОНИ СИСТЕМИ И ТЕХНОЛОГИЈЕ
ПОСЛОВНА ИНТЕЛИГЕНЦИЈА

ЗАВРШНИ РАД

АПРОКСИМАТИВНИ АЛГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ

Јаков Петровић 2017/3029

Београд 2020.

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

МАСТЕР АКАДЕМСКЕ СТУДИЈЕ

Сагласност чланова комисије за одбрану

Попуњавају чланови Комисије за одбрану:

Комисија која је прегледала рад
кандидата ПЕТРОВИЋ (ГОРАН) ЈАКОВ
под насловом АПРОКСИМАТИВНИ АЛГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ и одобрила одбрану:

Ментор: др Милош Јовановић, доцент

_____

Члан: др Гордана Савић, ванредни професор

_____

Члан: др Милан Вукићевић, ванредни професор

_____

БИОГРАФИЈА

Јаков Петровић је студент мастер студија на Факултету организационих наука. Члан је тима који је успешно представљао Факултет на такмичењу Америчког института за операциона истраживања и менаџмент (*INFORMS*) током 2018. и 2019. године. Био је међу одабраним студентима који су похађали летњу школу машинског учења 2018. која је организована у сарадњи Факултета организационих наука и Берлинске школе за економију и права. Основне студије на Факултету организационих наука уписао је 2013. и завршио у року одбранивши рад под називом „Мрежна апликација за симулацију аукција у којима сви плаћају“.

Основно и средњошколско образовање стекао је у Врњачкој Бањи. Као члан еколошких и друштвених организација учествовао на волонтерским пројектима у Србији, Норвешкој и Русији.

Запослен је на позицији „Сарадник за базе података са познавањем логистике“ у предузећу *Robert Bosch d.o.o.*, а у претходно радно искуство убрајају се позиције у банкама *Raiffeisen banka a.d.* и Сбербанк Србија а.д.

Поред енглеског, течно говори руски језик.

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

МАСТЕР АКАДЕМСКЕ СТУДИЈЕ

Изјава о академској честитости

Попуњава кандидат:

Петровић Горан Јаков	2017/3029
Презиме, име једног родитеља, име	Број индекса

Студијски програм: Информациони системи и технологије

Модул: Пословна интелигенција

Аутор завршног рада под насловом: АПРОКСИМАТИВНИ АЛГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ

Чија је израда одобрена на Седници Већа студијских програма мастер академских студија одржаној: 18. јуна 2018.

Потписивањем изјављујем:

- Да је рад искључиво резултат мог сопственог истраживачког рада;
- Да сам рад и мишљења других аутора које сам користио у овом раду назначио или цитирао у складу са Упутством;
- Да су сви радови и мишљења других аутора наведени у списку литературе/референци који су саставни део овог рада и писани су у складу са Упутством;
- Да сам довио све дозволе за коришћење ауторског дела који се у потпуносзи/целости уносе у предати рад и да сам то јасно навео;
- Да сам свестан да је плагијат коришћење туђих радова у било ком облику (као цитата, парафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, веб сајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторским и сродним правима, Службени гласник Републике Србије, бр. 104/2009, 99/2011, 119/2012), као и других закона и одговарајућих аката Универзитета у Београду и Факултета организационих наука;
- Да сам свестан да плагијат укључује и представљање, употребу и дустрибуирање рада предавача или других студената као сопствених;
- Да сам свестан последица које код доказаног плагијата могу проузроковати на предати завшни мастер рад и мој статус;
- Да је електронска верзија завршног рада идентична штампаном примерку и пристајем на његово објављивање под условима прописаним актима Универзитета и Факултета.

Београд, _____

Потпис студента _____

АПСТРАКТ

Алгоритам најближих суседа припада најједноставнијим предиктивним алгоритмима машинског учења. Класификацију или регресију алгоритам врши на основу скупа најближих познатих инстанци. Претрага најближих суседа (енг. *Nearest neighbour search*) је рачунски најзахтевнија фаза алгоритма, па утиче на скалабилност при раду са већим количинама података. На скалабилност не утиче само количина података, већ и број димензија које поседују. Алгоритам најближих суседа није једини који се заснива на претрази, то су и алгоритми кластеровања, системи препоруке и препознавања образаца, технике компресије и друге, што ову тему чини веома значајном. Како би се побољшале перформансе алгоритма, уводе се апроксимативне методе претраге суседа. Одабране апроксимативне технике биће представљене и анализиране у раду. Експерименталним путем тестираће се перформансе апроксимативних алгоритама над скуповима података под различитим параметрима и извести препоруке за њихово коришћење.

Кључне речи: апроксимативни *k-NN*, претрага најближих суседа, апроксимативна претрага најближих суседа, машинско учење, претрага сличности

ABSTRACT

The algorithm of k-nearest neighbors is among the simplest predictive algorithms of machine learning. The algorithm performs classification and regression based on a set of the nearest known instances. Nearest neighbor search is the most computationally demanding phase of the algorithm, so it affects scalability when working with larger amounts of data. Scalability is affected not only by the amount of data, but also by the number of dimensions they possess. There are many algorithms that rely on nearest neighbour search, such as clustering algorithms, recommendation and pattern recognition systems, compression techniques and others, which makes this topic important. Approximate neighbor search methods are introduced in order to improve the performance of the algorithm. Selected approximate techniques will be presented and analyzed in the paper. The performance of approximate algorithms over data sets under different parameters will be tested experimentally and recommendations for their use will be reported.

Key words: approximate kNN, nearest neighbour search, approximate nearest neighbour search, machine learning, similarity search

АКРОНИМИ

Annoy	*Approximate nearest neghbors Oh Yeah*
FALCONN	*Fast Lookups of Cosine and Other Nearest Neighbors*
flann	*Fast Library for Approximate Nearest Neighbors*
HNSW	*Hierarchial navigable small world graph*
k-D tree	*k-D* стабло (енг. *k-D tree*)
k-NN	к-најближих суседа (енг. *k-Nearest Neighbours*)
LSH	*Locality sensitive hashing*
MRPT	*Multiple random projection trees*
Nmslib	*Non-metric space library*
RP-tree	*Random projection tree*
SIFT	*Scale Invariant Feature Transform*
vp-Tree	*Vintage point tree*

СЛИКЕ

Слика 1: Апроксимативни алгоритам 4
Слика 2: *k-D* подела простора 8
Слика 3: *k-D* Бинарно стабло 8
Слика 4: *HNSW* структура (Malkov, A, & Yashunin, 2018) 12
Слика 5: Пример хеширања 14
Слика 6: Одзив и време претраге *SIFTSmall* 21
Слика 7: Одзив и конструкција индекса 22
Слика 8: *HNSW* - хеуристике 23
Слика 9: *Annoy*- утицај параметара 24
Слика 10: *SIFT1M* - почетни резултати 25
Слика 11: Одзив и Ts над *SIFT1M* на DataBricks 26
Слика 12: Време претраге и одзив на *SIFT1M* 27
Слика 13: Меморијски (лево) и рачунски захтеви (десно) 28
Слика 14: *Annoy* - Број стабала и перформансе 29
Слика 15: *Annoy*- раслојавање 30
Слика 16: *Annoy*- маргинални одзив и време претраге 30
Слика 17: *Annoy* - маргинални одзив и одзив 31
Слика 18: *LSH* - утицај параметара на перформансе 32
Слика 19: *LSH*- меморија 32
Слика 20: *HNSW* - одзив и време претраге 33
Слика 21: *HNSW- Tp* 34
Слика 22: *HNSW*- меморија 34

ТАБЕЛЕ

Табела 1: Подаци 17
Табела 2: Алгоритми 19
Табела 3: Параметри на *SIFT1M DataBricks* 26
Табела 4: *SIFT1M* на *DataBricks* - издвојени резултати 27
Табела 5: *FashionMINST* - резултати 35

ПРИЛОЗИ

Прилог 1:Почени резултати на *SIFT1M* подацима 42
Прилог 2: *Annoy* - меморијски захтеви 43
Прилог 3: *LSH* - Тр 43
Прилог 4: *Annoy* - Тр 43
Прилог 5:*SIFT1M* - Корелација 44
Прилог 6: *Annoy* - код за анализу параметара 44

Садржај

1. Увод 1
2. Дефиниција проблема и преглед стања у области 3
3. Апроксимативне технике претраге суседа 7
3.1 Подела простора и стабла 7
3.1.1 *k-D* стабла 7
3.1.2 *VP*- стабла 10
3.1.3 Друге методе поделе 11
3.2 Графовски приступ 11
3.3 Приступи засновани на хеширању 14
4. Анализа метода 15
4.1 Скупови података, индикатори перформанси и библиотеке 16
4.2 Селекција алгоритама 21
4.3 Анализа параметара 28
4.3.1 *Annoy* 29
4.3.2 *LSH* 31
4.3.3 *HNSW* 33
4.4 Анализа утицаја на предвиђање 34
4.5 Дискусија и препоруке 36
5. Закључак 38
6. Референце 39
7. Прилози 42

1. Увод

Рад се бави истраживањем апроксимативних техника за налажење најближих суседа над већим скуповима података. Претрага најближих суседа у метричком простору, подразумева проналажење инстанци, које су на најмањој удаљености од инстанце претраге.

Претрага најближих суседа је примењена у многим областима попут биомедицине, генетике, маркетинга, социјалних медија, машинског учења и оптимизације, користи се при обради слика и снимака, препознавању образаца и облика, те анализи и обради просторних и геолошких података. У већини случајева, записи из ових база се могу представити у векторском простору који има неколико десетина, па чак и хиљаду димензија. (Indyk & Motwani, 1998).

Алгоритам *k-NN* (енг. *k-nearest neighbours*) је предиктивни алгоритам машинског учења, који непознату класу или вредност инстанце, одређује на основу особина из скупа најближих познатих инстанци и широко је коришћен метод класификације и регресије.

Предвиђање будућих догађаја је један од кључних изазова са којима се суочавају друштва, предузећа и појединци. У складу са предвиђањима, ентитети могу предузети акције у циљу повећања прихода или смањења штете.

Добијање благовремених резултата применом алгоритама машинског учења, уз ефикасно коришћење ресурса, је императив. Једноставност је врлина *k-NN* алгоритма, али може бити узрок слабијих перформанси код података са шумовима, или кад је потребно обрадити велики број димензија и инстанци.

Претрага суседа је кључна и временски критична фаза алгоритма. Из претходно наведеног, намеће се закључак да су класификација и регресија завршне и једноставне фазе, које следе захтеван и сложен проблем претраге најближих суседа (енг. *Nearest neighbour search*). Зато проблем брзог налажења суседа представља предмет овог рада. Укорењено је мишљење да ће линеарна, односно егзактна претрага, увек бити спорија од апроксимативне претраге. Линеарна претрага врши пролазак кроз цео скуп података, то јест захтева рачунање удаљености између тачке претраге и свих осталих инстанци скупа, при том водећи рачуна о *k* њближих. Овакав приступ се често назива приступом

грубе силе (енг. *brute force*) или исцрпном претрагом (енг. *exhaustive search*). Апроксимативна претрага подразумева проналажење суседа који могу, али не морају бити најближи суседи инстанце претраге. Најзначајнија особина апроксимативног алгоритма је да постигне готово исту прецизност као оригинални приступ са линеарном претрагом, али за значајно краћи временски интервал и не захтева пролазак кроз све инстанце скупа.

Управо је та чињеница мотивисала су аутора рада на изучавање апроксимативних метода претраге суседа. Резултат истраживања ће приказати рад апроксимативних алгоритама и пружити преглед и анализу њихових параметара. Утврдиће се разлика у перформансама апроксимативних техника над скуповима података различитих величина и димензија, на корист будућим истраживачима.

У другом поглављу је дефинисан појам претраге најближих суседа, појам апроксимативне претраге суседа и представљен је досадашњи напредак у области. У трећем се даје опис најзначајнијих представника апроксимативних метода, које су инспирација новим, измењеним или побољшаним верзијама чије се перформансе тестирају у наредним поглављима. Утицај параметара на перформансе селектованих алгоритама предмет је четвртог поглавља, у ком се такође формулишу савети за одабир алгоритама и подешавање њихових параметара.

2. Дефиниција проблема и преглед стања у области

У формалном облику, проблем налажења најближег суседа се састоји у следећем:

Нека је дат скуп тачака $P = \{ p_1, p_2, \dots, p_n \}$ у простору R^d. Потребно је претражити скуп P, како би се нашла тачка која је најближа задатој тачки $q \in R^d$:

$$\underset{p \in P}{arg\,min}\, dist(q, p)$$

где је $dist(q, p)$ функција раздаљине метричког простора.

За разлику од алгоритама који испрва изводе генерализацију података и генеришу класификациони модел, те на основу модела брзо процењују припадност непознате инстанце одређеној класи (енг. *eager learners*), овај метод припада групи лењих алгоритама (енг. *lazy learners*). Одлика лењих алгоритама је да се инстанце просто складиште у фази тренирања, или се извршава минимално претпроцесирање података. То значи да ће фаза тренирања алгоритма бити краћа, међутим предвиђање алгоритма ће трајати дуже. Како се фаза тренирања код ових алгоритама састоји у простом чувању инстанци, ова група алгоритама се назива и инстанцно-оријентисаном. Овај алгоритам је често веома рачунски интензиван и захтева ефикасне технике складиштења података, које треба да омогуће паралелно извршавање поступка проналажења суседа (Han, Pei, & Kamber, 2012).

Иако је проблем налажења суседа наизглед једноставан, постоје два изразита проблема у вези са скалабилношћу при раду са великом количином података:

- Време извршавања: Алгоритамска сложеност проналажења само једног најближег суседа је $O(nd)$, где је n број тачака у простору и d број димензија. Алгоритам је рачунски захтевнији када је потребно пронаћи више суседа, што је најчешћи случај. Најзад, цео процес је потребно извести сваки пут када се појави инстанца коју треба класификовати.
- Меморија: За брзо рачунање удаљености, неопходно је да је цео скуп постојећих тачака учитан у *RAM* меморију, што може представљати изазов код великих база података. (Maillo, Ramirez, Triguero, & Herrera, 2017)

Ефикасно претраживање најближих суседа постаје учестали захтев према великом броју база података. Алгоритми који се директно ослањају на индексне структуре, дају задовољавајуће резултате на малим или средњим количинама података, али не

задовољавају потребе ефикасности када је реч о великим и високо димензионираним скуповима. (Seidl & Kriegel, 1998).

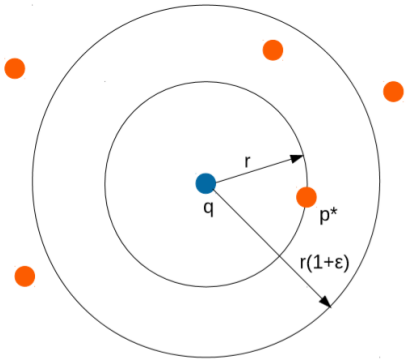
Очигледна тешкоћа да се осмисле алгоритми засновани на претрази суседа који ће бити ефикасни у случају када је број димензија и инстанци већи (код неких аутора када је број димензија већи од 10), наводи да је потребно размотрити апроксимативне методе које ће проналазити приближно најближе суседе.

У том случају, може се проширити претходно дата дефиниција:

Нека је дат скуп тачака P у простору R^d и нова тачка $q \in R^d$ којој треба одредити суседа. За дато $\varepsilon > 0$, тачка $p \in P$ је $(1+\varepsilon)$ приближан сусед ако је:

$$dist(q,p) \leq (1+\varepsilon)dist(q,p^*)$$

Где је p^* најближи сусед задате тачке q. У општем случају, када је $1 \leq k \leq n$, поступак се своди на проналажење k приближних најближих суседа, где за сваког i - тог важи да је апроксимација одговарајућег најближег суседа тачке q (Arya, Mount, Netanyahu, Silverman, & Wu, 1998).



Слика 1: Апроксимативни алгоритам

Важно је напоменути да се међу имплементацијама апроксимативне претраге, ретко налазе оне, које дају гаранције да ће се пронађен сусед наћи у $(1+\varepsilon)$ околини најближег суседа.

Проблем налажења суседа био је у пољу интересовања истраживача од педесетих година двадесетог века, међутим, област долази у фокус почетком деведесетих година када је објављено мноштво значајних радова из ове области. Према наводу (Kushilevitz, Ostrovsky, & Rabani, 2000), када је реч о веома димензионираним подацима, проблем је разматран од стране Добкина и Липтона (*Dobkin, D., & Lipton, R. J. (1976).*

Multidimensional searching problems) који су генерисали алгоритам чија сложеност претраге експоненцијално зависи од броја димензија d. Представљени метод је касније побољшан, али је време извршења још увек било у експоненцијалној зависности од димензије простора.

Препоруке за избор структуре погодне за претрагу у зависности од карактеристика података који се обрађују дали су (Muja & Lowe, 2009). Утврђено је да два алгоритма од којих се оба заснивају на стаблима (енг. *randomized kd-tree* и *hierarchical k-means tree*) пружају задовољавајуће резултате. Истраживања су показала да би број инстанци код оваквих структура, морао да буде значајно већи у односу на број димензија $n \gg d^2$, јер у супротном не би било побољшања у времену у односу на линеарну претрагу свих инстанци (Arya, Mount, & Narayan, Accounting for boundary effects in nearest-neighbor searching, 1996)

Крајем века, (Kleinberg, 1997) је представио два алгоритма који дају значајна побољшања у односу на дотадашња достигнућа. Први алгоритам нуди сложеност претраге од $O\big((d^2\ log^2 d)(d + logn)\big)$ и други, чија сложеност тежи линеарној (Kleinberg, 1997). Алгоритми подразумевају пројекцију вектора у простор са нижим бројем димензија и креирање графовске структуре.

Другачији приступ решавању проблема донели су (Indyk & Motwani, 1998) и објавили метод који има полиномну сложеност у зависности од $\log n$ и d , дакле $O(d\ poly\ \log n)$. Идеја је да се пронађе метод којим ће се међусобно слични објекти сврставати у исте категорије. Хеширање на основу положаја, или локацијски сензитивно хеширање (енг. *Locality-sensitive hashing, LSH*) је назив представљене технике. Формулација коју даје (Samet, 2006) пружа боље објашњење:

Циљ *LSH* алгоритма је да се пронађе функција хеширања која ће сачувати информацију о удаљености инстанци, или приближне удаљености уз одређену толеранцију.

Општи кораци *LSH* алгоритма представљени су у раду под називом „Претраживање сличности при великом броју димензија путем хеширања“ (Gionis, Indyk, & Motwani, 1999).

Већина до данас представљених решења, подразумева претпроцесирање постојећих инстанци, како би се над њима касније извршила претрага.Читав скуп апроксимативних метода предвиђа преуређење скупа података, тако што би се формирала структура стабла (*k-D tree, R-tree, VP-tree, SA-tree* и друга), чиме се обезбеђује убрзање алгоритма.

Друге методе предлажу хеширање инстанци како би се спровело груписање међусобно сличних, или пак организовање података у графовску структуру, где би блиске инстанце биле повезане гранама графа.

При одабиру апроксимативног алгоритма, пожељно је размотрити време, сложеност, као и меморијске захтеве обраде података. Велики број аутора даје одличан теоријски допринос, међутим, недостатак многих радова огледа се у изостанку свеобухватне експерименталне анализе. Понекад недостаје програмска имплементација алгоритма, или се имплементација изводи искључиво на мањим и вештачки креираним скуповима података, а понекад изостаје и упоредна анализа са изворним алгоритмом који се ослања на комплетну претрагу података.

Новија истраживања представљају решења проблема апроксимације ослањајући се на технологије за паралелизацију процеса и дистрибуиране рачунарске архитектуре. Неколико дистрибуираних алтернатива су предложене са циљем да омогуће апроксимативном алгоритму да обради велику количину података. Већина решења се базира на програмској парадигми *MapReduce* и њеној имплементацји отвореног кода – *Hadoop*. На тај начин се паралелизује извршавање алгоритма и ублажавају ефекти меморијских и рачунских ограничења. Недавно је представљен нови дизајн (Maillo, Ramirez, Triguero, & Herrera, 2017) који превазилази стандардне *Hadoop-MapReduce* приступе и обезбеђује флексибилну шему за класификацију великог броја непознатих инстанци над великим подацима, користећи *Apache Spark* архитектуру (Triguero, Maillo, Luengo, García, & Herrera, 2016).

Поменути радови објављени у прошлом веку инспирисали су истраживаче да осмисле нова, или дају значајна побољшања постојећих метода. Сва досадашња решења за апроксимативну претрагу суседа могу се оквирно сврстати у четири групе:

1. Приступи засновани на подели простора и стаблима
2. Графовски приступ
3. Приступ заснован на хеширању
4. Приступ заснован на векторској квантизацији

Подела је грубо дефинисана и могуће је наћи примере који користе мешовите приступе, попут (Douze, Sablayrolles, & Jégou, Link and code: Fast indexing with graphs and compact regression codes, 2018) који комбинују графовске технике са приступима векторске квантизације. У наредним поглављима дати су описи неких представника ових група, који су имплементирани у програмским библиотекама отвореног кода.

3. Апроксимативне технике претраге суседа

У претходном поглављу је пружен осврт на правце истраживања у области апроксимативне претраге суседа. У поглављима која следе даје се детаљан опис најзначајнијих представника класа. Објашњени су поступци претраге суседа и креирање структура података над којима се извршавају. Представљена су два егзактна алгоритма, *k-D* и *VP* стабла, који уз увођење додатних ограничења постају апроксимативни. Међу апрокимативним методама издвајају се *HNSW* који се заснива на графовима и *LSH* који је заснован на хеширању. Теоријски приказ сложености креирања структуре и претраге алгоритама омогућава тумачење резултата истраживања и олакшава одабир метода за тестирање.

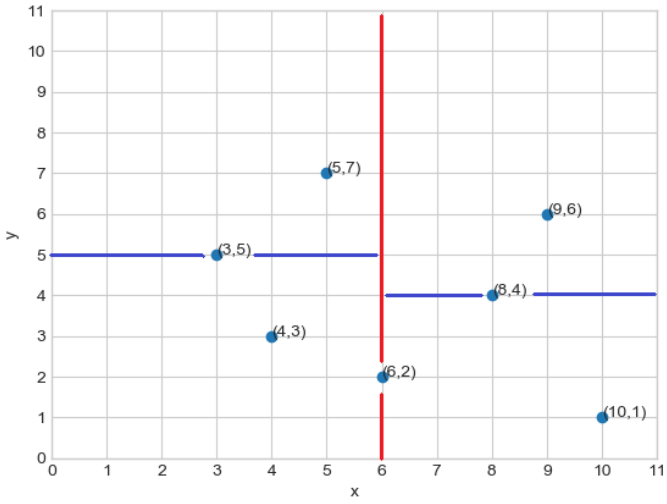
3.1 Подела простора и стабла

Како би убрзали поступак налажења суседа и смањили обим претраге, истраживачи су се окренули техникама за поделу простора. У највећем броју случајева, ради се о бинарној подели, која подразумева рекурзивно дељење на два дела, све док је испуњен критеријум поделе. Најмањи подпростор добијен овим путем се често назива ћелија. Бинарно стабло је структура података која је погодна за организацију елемената из тако подељеног простора. Тачке сваког дела, постају елементи левог и десног подстабла, респективно. Исправно је претпоставити да ће се најближи суседи наћи у ћелији којој припада тачка претраге, или пак у суседним ћелијама. Један од најбитинијих фактора по којима се разврставају алгоритми ове класе, је метод поделе простора. Код најпознатијег представника групе, *k-D* стабла, простор у сваком кораку дели хиперраван која је нормална на осу неке димензије. Представници ове групе су *RP* стабла, где је хиперраван насумично одређена , *2-means* стабла где се подела спроводи на основу алгоритма кластеризације, *PCA*-стабло које простор дели на основу анализе главних компоненти, *VP*-стабло и друга.

3.1.1 *k-D* стабла

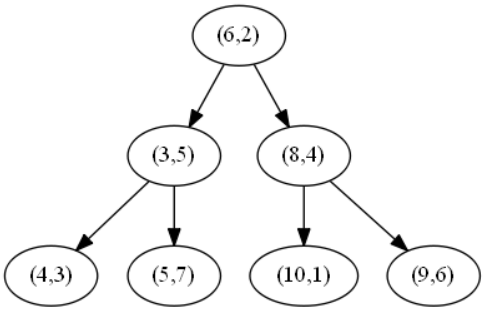
k-D стабло је један је од првих и најпознатијих представника групе. Посматрајмо једноставан пример дводимензионалног простора ($k = 2$). У основном облику, алгоритам налази медијалну вредност међу координатама одабране осе, у овом случају

x осе. Медијална вредност ће дефинисати хипераван поделе. У случају када имамо две димензије, хипераван ће заправо бити права, која дели простор на два дела.



Слика 2: *k-D* подела простора

Како је медијална вредност по x оси једнака 6, у првој итерацији, простор је подељен у односу на тачку са координатама (6,2) - која постаје корен стабла. Остале тачке из простора постају чворови левог или десног подстабла, у зависности од положаја у односу на праву $x = 6$. У наредној итерацији простор се дели у односу на y осу, за сваки претходно добијен подпростор. Праве $y = 5$ и $y = 4$ деле леви и десни подпростор, па тачка (3,5) постаје корен левог подстабла, а тачка (8,4) десног. Бинарно стабло добијено на овај начин представљено је на Слика 3: *k-D* Бинарно стабло.



Слика 3: *k-D* Бинарно стабло

Алгоритам је лако проширити на случај када је број димензија k већи ($k > 2$) . У свакој итерацији простор се дели у односу на димензију k. Услов за заустављање поделе може бити дат кроз ограничење броја нивоа које стабло може имати, дефинисањем броја елемената који ће се наћи у листовима, или на неки други начин.

k-D је егзактан метод, што значи да ће резултат претраге бити оне инстанце које су прави суседи тачке претраге. Из тог разлога, алгоритам не проверава само садржај ћелије којој припада тачка претраге, већ и суседних ћелија уколико је потребно (Алгоритам 1). Вратимо се на дводимензионални пример. Нека тачка претраге има координате (7,9). Претрага започиње од корена стабла који се проглашава најближим. Како корени чвор није лист у стаблу, услов из 2. линије није задовољен. Алгоритам се наставља у десном подстаблу, јер је координата тачке претраге већа од осе поделе простора (линија 9). Рекурзивним позивима долази се до чвора (9,6), који се налази у ћелији којој припада тачка претраге и који се проглашава најближим. Претрага се поново наставља у кореном чвору где се утврђује да је дистанца између тачке претраге и тачке (9,6), већа од удаљености од осе поделе (линија 11), па постоји могућност да се најближи сусед налази у левом подстаблу. На исти начин, алгоритам пролази кроз лево подстабло и налази правог најближег суседа (5,7).

Алгоритам 1. Претрага у *k-D* стаблу

```
1:   pretraga(q, n, p, d)
2:   if  n.l = ∅  ∧ n.r = ∅ then return p,d;
3:   dc ≔ dist(q,n)
4:   if dc < d then p ≔ n; d≔dc;
5:   if  n.l ≠ ∅  ∨ n.r ≠ ∅
6:     if q.ax ≤ n.ax
7:       If q.ax – d ≤ n.ax then p,d ≔ pretraga(q, n.l, p, d)
8:       If q.ax + d > n.ax then p,d ≔  pretraga(q, n.r, p, d)
9:     if  q.ax > n.ax
10:      If q.ax + d > n.ax then p,d ≔ pretraga(q, n.r, p, d)
11:      If q.ax – d ≤ n.ax then p,d ≔  pretraga(q, n.l, p, d)
```

Читалац ће приметити да се егзактан метод лако може претворити у апроксимативни, тако што ће се ограничити број чворова који се посећује, односно спречити претрага суседних ћелија. У повољном случају, сложеност претраге *k-D* стабла је $O(\log n)$, али се показало да сложеност може бити већа од линеарне, што су потврдила и емпиријска истраживања. Метод је показао неефикасност када је број димензија већи од 12 (Marimont & Shapiro, 1979).

3.1.2 VP- стабла

Једна од најпознатијих подврста Лопта-стабала (енг. *Ball-tree*) је *VP*-стабло (енг. *Vantage point tree*). Код креирања стабла, испрва је потребно из скупа тачака $P = \{p_1, p_2, \ldots, p_n\}$ изабрати тачку која ће бити пивот - p_t. Потом се израчунава медијална удаљеност r између пивота и осталих тачака, па се скуп дели на два дела:

$$P_1 = \{p \in P \,/\, p_t | d(p_t, p) < r\}$$

$$P_2 = \{p \in P \,/\, p_t | d(p_t, p) \geq r\}$$

тако да су тачке из скупа P_1 унутар полупречника r, док су остале тачке ван овог полупречника. Рекурзивном применом овог правила гради се бинарно стабло где су унутрашњи чворови пивоти, чији су елементи левог и десног подстабла унутар и изван одговарајуће лопте (Samet, 2006). Пивот се може изабрати насумично, или на неки други начин. Једна од предложених метода, од које се очекује да резултира креирањем стабла које пружа добре перформансе, дефинисао је (Yianilos, 1993). Идеја је да се из насумично одабраног подскупа тачака пронађе она, која има најбољу расподелу удаљености у односу на друге тачке. Такав метод форсира избор оне тачке која је удаљена од средине, односно скрајнута на ивицама скупа. Поступак креирања стабла може се дефинисати једноставним алгоритмом:

Алгоритам 2. Креирање *VP*-стабла

```
1:  kreirajVP(P)
2:  if P = ∅ then return ∅
3:  P1 = ∅
4:  P2 = ∅
5:  n := kreirajCvor()
6:  n.pt := izaberiPivot(P)
7:  n.med := Medijana(dist(n.pt, P))
8:  For each p ∈ P
9:      If dist(p, n.pt) < n.med then P1.add(p)
10:     If dist(p, n.pt) ≥ n.med then P2.add(p)
11: n.l = kreirajVP(P1)
12: n.l = kreirajVP(P2)
11: return n
```

Претрага се врши на сличан начин као код *k-D* стабала, па и овде важи аналогија да се апроксимативна варијанта алгоритма може добити ограничавањем броја чворова који процедура испитује.

3.1.3 Друге методе поделе

Посматрајмо тачке скупа P као векторе. За конструкцију *RP*-стабла, испрва се насумично бира вектор v (често јединични вектор) , потом се врши пројекција свих вектора из P на v , да би се на крају одредила медијална пројекција и скуп поделио на два дела.

$$P_1 = \{p \in P | p \cdot v \leq med(p \cdot v)\}$$

$$P_2 = \{p \in P | p \cdot v > med(p \cdot v)\}$$

У дводимензионалном простору, геометријском смислу, простор би делила права која је ортогонална на насумично дефинисану праву. Ортогонална права дели простор на делове са једнаким бројем тачака.

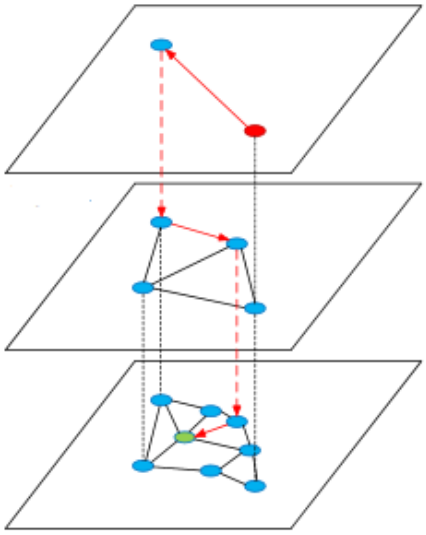
У раду ће се тестирати и *k-means* стабло које се формира рекурзивним кластеровањем података и *randomised k-D tree* које се формира насумичним одабиром димензије по којој се врши подела код *k-D* стабала.

3.2 Графовски приступ

Најзначајнији приступ који је недавно објављен и пружа обећавајуће резултате је *HNSW* и се сврстава у алгоритме који користе графовске структуре за проналажење најближих суседа у метричком простору. Тачке у простору представљају чворове графа, а неусмерене гране су везе међу суседима. Хеуристике које врше претрагу структуре користе се и при додавању нових чворова. Наиме, одабрана хеуристика претражује структуру у потрази за најближим суседима, када су услови претраге задовољени, елемент постаје чвор графа са гранама које га везују за суседе. Алгоритам је дефинисан у (Malkov, A, & Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018).

Аутори представљају хијерархијски организовану структуру графова, за коју тврде да има логаритамску сложеност претраге. Главни чиниоци који доприносе логаритамској сложености су, експлицитна селекција улазног чвора у графовску струкуру и напредне хеуристике за брзо налажење суседа у оквиру графа. Структура се назива хијерархијском јер је организована у више међусобно повезаних нивоа. На свим нивоима хијерархије формира се граф, са тим што се број чворова који формирају граф смањује са порастом нивоа. На најнижем нивоу, све тачке векторског простора формирају граф. На сваком наредном нивоу, граф ће формирати подскуп тачака из претходног нивоа. Формалан опис претходно описане структуре може се дати на следећи начин:

Нека је дат скуп тачака P векторског простора R^d и нека је дат скуп нивоа хијерархије $L = \{l_i | i = 0, \dots, k\}$. Све тачке скупа P биће организоване у графовску структуру на најнижем нивоу хијерархије l_0 те може се записати $P = P_{l_0}$. За скуп тачака P_{l_1} које су саставни део графа на нивоу l_1 важи да је $P_{l_1} \subset P_{l_0}$. У општем случају $P_{l_i} \subset P_{l_{i-1}}$.



Слика 4: *HNSW* структура (Malkov, A, & Yashunin, 2018)

Из оваквог описа, јасно је да ће се поједине тачке векторског простора јављати на неколико нивоа хијерархије. Важно је напоменути да међу нивоима структуре постоје везе. Тачка која представља чвор графа на нивоу l_i имаће референцу на чвор исте те тачке на нивоу l_{i-1}.

Графови се конструишу итеративним додавањем елемената из скупа P, тако што сваки постаје чвор графа и повезује се са M најближих суседа. Испрва се за сваки елемент који се додаје у структуру одреди максимални ниво на којем ће се појавити l_{max}.

Вероватноћа да се елемент нађе на високим нивоима је експоненцијално опадајућа, чиме се обезбеђује мали број чворова у вишим слојевима хијерархије. Процес додавања у структуру се може поделити у две фазе:

- Проналажење улазних чворова, почевши од највишег нивоа l_k па до максималног дефинисаног нивоа на ком се чвор додаје l_{max}
- Проналажење најближих суседа на максималном нивоу за елемент и његовим поднивоима и повезивање са њима ($l_{max}, \dots, l_0$)

Као што је напоменуто, хеуристика која служи за претрагу нивоа користи се у обе фазе. Разлика је што се у првој фази параметри постављају тако да се проналази само један најближи сусед.

Прва фаза додавања започиње у највишем нивоу где се проналазе најближи суседи за елемент који се додаје. Најближи сусед пронађен у највишем нивоу, постаје улазни чвор за претрагу на нижем нивоу, где се поступак претраге понавља. Суседи улазног чвора који су ближи елементу који се додаје у граф постају кандидати за најближег суседа. Како је у овој фази потребно наћи најближег суседа елемента који се додаје, хеуристика „прождрљиво“ бира само једног кандидата из листе суседа. У наредном кораку посматрају се суседи одабраног кандидата. На овај начин хеуристика у сваком чвору бира суседа који задовољава критеријум да је најближи елементу који се додаје. Претрага се завршава оном чвору, за којег важи да нема суседе који би били ближи. Тада се прелази на нижи хијерархијски ниво.Друга фаза започиње када алгоритам спусти на ниво хијерархије где ће елемент први пут постати чвор графа.

Ако би параметри алгоритма били подешени тако да постоји само један ниво, алгоритам би се свео на нехијерархијску верзију дефинисану у (Malkov, Ponomarenko, Logvinov, & Krylov, 2014). Како би се избегло да се алгоритам претраге заустави у неком локалном минимуму, аутори су саветовали да се претрага започне у чворовима који имају висок степен повезаности (енг. *hubs*). Како је број високо повезаних чворова растао са повећањем броја елемената, претрага је тежила полилогаритамској сложености. Увођењем концепта хијерархије, очекивало се смањење сложености претраге и вероватноће заустављања у локалном минимуму, што је чест случај код кластеризованих података. Наведене претпоставке су се потврдиле у досадашњим експериментима. Аутори наводе да је фаза креирања графа погодна за паралелизацију.

3.3 Приступи засновани на хеширању

Хеш функције пресликавају податке произвољне дужине у податке фиксне дужине – често је податак фиксне дужине у облику бинарног записа. Ситуација у којој два различита податка имају исту хеш вредност назива се колизија. Колизија је неминовна појава код хеширања и често је непожељна у области криптографије. Код проблема претраге сличности, колизија је веома пожељна! Тада два различита, али међусобно слична података добијају исту хеш вредност. Циљ локацијски сензитивног хеширања (*LSH*), је да додели исту хеш вредност тачкама које су близу једна другој. Концепт су формулисали (Indyk & Motwani, 1998).

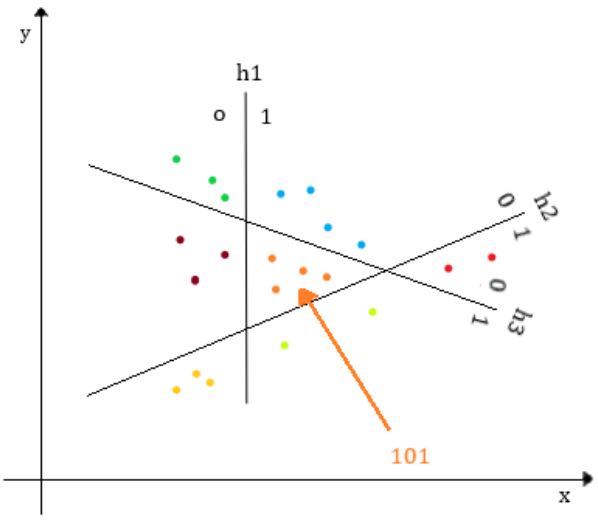
Нека је дат простор S . Фамилија хеш функција $H = \{h : S \rightarrow U\}$ је (r, cr, p_1, p_2) сензитивна за функцију удаљености $dist(p, q)$ за свако $p, q \in S$ када важи да:

$$\text{ако } dist(p, q) \leq r \text{ онда } Pr_H[h(p) = h(q)] \geq p_1$$

$$\text{ако } dist(p, q) \geq cr \text{ онда } Pr_H[h(p) = h(q)] \leq p_2$$

Дакле, ако је удаљеност међу тачкама мања или једнака r, вероватноћа њихове колизије треба да буде већа или једнака p_1. Када је удаљеност већа од cr, вероватноћа колизије мора бити мања или једнака p_2.

У духу претходно наведених техника поделе простора, може се рећи да две тачке које су близу једна другој, треба да припадају истој партицији простора са вероватноћом не мањом од p_1. У том случају хеш функције би дефинисале неку хиперраван за поделу.



Слика 5: Пример хеширања

Помоћу фамилије хеш функција, може се конструисати индексна структура, на следећи начин:

За цео број M, дефинишимо фамилију функција $G = \{g : S \rightarrow U^M\}$ такву да за $\forall g \in G$ важи да је $g(h_1(v), \dots, h_M(v))$, где $h_j \in H$ и $1 \leq j \leq M$. Овде је g заправо конкатенација M хеш функција.

За цео број L, насумично изаберимо функције $g_1, \dots, g_L$ из G. Свака појединачна функција $g_i, 1 \leq i \leq L$, користи се за конструисање појединачне хеш табеле. Дакле, креира се L хеш табела.

У складу са претходно наведеним, потребно је:

1. Извршити иницијализацију L хеш табела
2. За сваку табелу генерисати M хеш функција
3. Унети тачке v рачунајући хеш вредност $g_i(v)$ за сваку табелу L_i.

Тачке којима је додељена иста хеш вредност у табели припадају истој корпи (енг. *bucket*). Претрага се врши тако што се се генерише хеш вредност тачке претраге за сваку табелу. Потом се саставља листа кандидата из корпи свих табела којима припада тачка претраге. Кандидати се напослетку сортирају по удаљености и извлачи се k најближих . Да би резултати претраге били смислени, потребан је велики број табела, који за последицу има велике меморијске захтеве. Овај проблем би се могао решити на уштрб времена претраге, тако што би се у свакој табели проверавало више од једне корпе. Алгоритам *Multi-Probe LSH*, решава проблем тако што у свакој табели проналази „суседне“ корпе и додаје у листу кандидата.

4. Анализа метода

Анализа перформанси апроксимативних техника за претрагу суседа представљена је у овом поглављу. У истраживању су коришћене библиотеке са постојећим имплементацијама претходно описаних алгоритама. Уобичајено је за овакве библиотеке да се изворни алгоритми делимично мењају, или комбинују ради побољшања претраге, или смањења рачунске сложености. На наредним страницама дат је опис кључних параметара за поређење техника и скупова података над којима се поређење врши.

Поглавље пред нама би могло да пружи одговоре на следећа питања:

- Које су најпопуларније апроксимативне технике?
- Колико је времена потребно за извршавање линеарне претраге и апроксимативне претраге?
- Колико се разликују резултати апроксимативне и линеарне претраге суседа?
- У којој мери је могуће паралелизовати извршење алгоритма?
- Колики је утицај структуре података на брзину извршавања апроксимативних варијанти алгоритма?
- Које ће перформансе и резултате апроксимативни алгоритми постићи при раду над скуповима различитих величина и броја атрибута?
- Да ли је подешавање параметара једноставно и у којој мери утиче на перформансе?
- Да ли је могуће извести препоруке за коришћење одговарајућег апроксимативног алгоритма у зависности од података и корисничких захтева?

Методолошки поступак почиње тестирањем на мањем скупу података на персоналном рачунару, како би се селектовали алгоритми који улазе у другу фазу, у којој се анализира утицај параметара на кључне перформансе. У поглављима која следе, детаљније се описују скупови података над којима се претрага врши, дефинишу се кључне мере перформанси алгоритама и библиотека којима припадају и спроводи дубља анализа одабраних алгоритама.

4.1 Скупови података, индикатори перформанси и библиотеке

Скупови података за тестирање најчешће садрже три дела: тренинг, тест и контролни скуп података. Скуп података за тренирање је онај у коме се врши претрага најближих суседа. Тест скуп садржи тачке претраге, односно оне инстанце за које је потребно одредити суседе. Контролни скуп (енг. *ground truth*), садржи правилна решења претраге, то јест садржи листу правих суседа из тренинг скупа.

Скупови података који се најчешће користе у референтној литератури, над којима се врши анализа и у овом раду, приказани су у табели:

Назив	Број димензија	Тренинг инстанце	Тест инстанце	Величина	Број суседа
SIFTsmall	128	10000	100	4.92 *MB*	100
SIFT1M	128	1000000	10000	492 *MB*	100
FashionMINST	784	60000	10000	217 *MB*	100

Табела 1: Подаци

Колоне имају следећа значења :

- Број димензија – величина векторског простора; Свака инстанца из скупа података је вектор са назначеним бројем димензија
- Тренинг инстанце – број инстанци, односно вектора у тренинг скупу података
- Тест инстанце – број вектора у тест скупу за које је потребно наћи најближе суседе из скупа података за тренинг
- Величина – меморијски простор који скуп заузима на диску рачунара
- Број суседа – показује колико је суседа потребно пронаћи за сваку инстанцу из тест скупа

Над овим подацима није уобичајено вршити чишћење, премда је извршена брза анализа, која неће бити представљена у раду - пре свега због потешкоћа да се дистрибуција, корелисаност и постојање кластера прикажу при великом броју атрибута и инстанци. Ипак, заинтересовани читалац може наћи визуелизацију материце корелације у прилозима (Прилог 5). Број суседа који се тражи једнак је за сваки скуп података и сваки алгоритам и износи једну стотину.

SIFT1M и *SIFTsmall* садрже векторе, чије димензије представљају дескрипторе фотографија који су погодни за претрагу. Подаци су први пут коришћени код (Jegou, Douze, & Schmid, 2010). Датотеке су у бинарном формату, а свака димензија је представљена тридесетдвобитним децималним бројем.

Популарни скупови података, понекад нису класификовани (*SIFT1M*), или је пак забрањено користити доступну класификацију због власничких права (нпр. *NewYorkTimes* скуп), па је над њима могуће проверити једино брзину претраге и број тачно идентификованих суседа. Брзину претраге, али и тачност предвиђања могуће је проверити на малом, али високо димензионираном скупу *FashionMINST,* који је класификован. Инстанце скупа представљају фотографије одевних предмета у

векторском запису и све се могу сврстати у десет категорија. Скуп података се састоји од шездесет хиљада слика.

Дефинисано је шест кључних показатеља перформанси. У квантитативне индикаторе убрајају се:

- Грешка удаљености (E) – разлика у вредности функције удаљености од непознате инстанце између апроксимативних и правих суседа
- Одзив (R) – број идентификованих правих суседа у односу на број тражених
- Време претпроцесирања (T_p) – процесорско време потребно за конструисање индексне структуре у секундама
- Време претраге (T_s) – процесорско време потребно за проналажење суседа у секундама
- Меморијски захтеви у *Gb*

Процесорско време указује на укупно време које је процесор утрошио на конструисање индекса и претрагу. У њега не спада време које је утрошено на друге процесе система. Оно међутим укључује утрошено време на свим процесорима код вишепроцесорских система, па се може догодити да буде веће од реално протеклог времена. У хипотетичком случају, за алгоритам који користи две нити, реално време извршавања може бити 5 секунди, али је у том случају укупно (процесорско) време 10 секунди.

Грешка удаљености се формално дефинише као количник просечне удаљености суседа и тачака претраге Q , на скупу апроксимативних суседа P и правих суседа P^*:

$$E = \frac{avg(dist(P,Q))}{avg(dist(P^*,Q))} \cdot 100\%$$

Одзив показује удео броја пронађених правих суседа:

$$R = \frac{|\ P\ \cap P^*|}{|P^*|}$$

Меморијски захтеви се мере у фази креирања индекса на удаљеном рачунару, користећи *psutil* библиотеку (Rodola, 2008) позивом методе *virtual_memory()*, која проверава заузетост меморије пре и након креирања индекса.

Једноставност подешавања параметара намеће се као квалитативни индикатор перформанси. Једноставно подешавање параметара подразумева да је за генерисање добрих квантитативних индикатора потребно подесити један параметар, или да је подешавање аутоматско. Средње тешко подешавање подразумева подешавање 2 параметара, док комплексно захтева сложене комбинације више параметара за добијање добрих резултата.

Над *FashionMINST* скупом је могуће тестирати и квалитет предвиђања апроксимативних алгоритама, па се као мере квалитета користе макро прецизност (P_m) и макро одзив (R_m), чије су дефиниције дате у поглављу 4.4.

Већина алгоритама је имплементирана у библиотекама отвореног кода у језицима *C* и *C++*. Постоје имплементације у другим програмским језицима као што је *Python*, али се оне ретко користе због слабијих перформанси тог програмског језика у односу на горе наведене. Коришћене библиотеке имају омотаче (енг. *wrappers, Python bindings*) у *Python* језику, што олакшава њихово коришћење, али отежава корекције самог алгоритма.

У разматрање се узимају следеће библиотеке са припадајућим алгоритмима:

Назив алгоритма	Библиотека	Концепт	Језик	Врста
HNSW	*Nmslib*	Граф	*C++*	Апр.
vp-tree	*Nmslib*	*VP-* стабла	*C++*	Апр./Екз.
Annoy	*Annoy*	*2-means tree forest*	*C++*	Апр.
mrpt	*MRPT*	*RP-tree forest*	*C++*	Апр.
rkdTree	*flann*	*randomized k-D*	*C++*	Апр.
kmeans	*flann*	*k-means tree*	*C++*	Апр.
linear	*flann*	Линеарна претрага		Егз.
k-D	*Scikit-learn*	*k-D*	*Python* и *Cyton*	Егз.
ball-tree	*Scikit-learn*	*Ball-tree*	*Python* и *Cyton*	Егз.
LSH	*FALCONN*	Хеш табеле	C++	Апр.

Табела 2: Алгоритми

Анализа започиње над мањим скупом података и врши се на персоналном рачунару са процесором од два језгра јачине 2×2.4 *Ghz* и 4*Gb* меморије, под *Windows* оперативним системом. Сложеније анализе се спроводе на удаљеном рачунару на *Databricks* платформи, са ограниченим ресурсима у погледу меморије, процесора и времена коришћења. Доступно је 15 *Gb* меморије и процесор од 3*Ghz* и максималним периодом мировања рачунара од 2 сата.

Библиотека *nmslib* је представљена у (Boytsov & Bilegsaikhan, 2013), а касније допуњена *HNSW* алгоритмом (Malkov, A, & Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018) и од 2020. је подржана од стране аналитичке платформе *Amazon Elasticsearch Service*. Развој библиотеке *Annoy,* која се користи се у системима препоруке, започео је Ерик Бернардсон у компанији за дистрибуцију музичког садржаја *Spotify* (Spotify Technology S.A., 2015). Јединa библиотека која намењена за *Linux* оперативни систем је *FALCONN,* па су њене могућности тестиране само на удаљеном рачунару. Развијена је на основу рада (Andoni, Indyk, Laarhoven, Razenshteyn, & Schmidt, 2015) од аутора поменутих у другом поглављу, који су познати по развоју *LSH* метода. Библиотека *flann* представља имплементације алгоритама дефинисаних у (Muja & Lowe, 2009), и садржи функције за аутоматско бирање алгоритма и њихових параметара. Позната библиотека у области машинског учења *Scikit-learn* је представљена у (Pedregosa, и други, 2011), а *MRPT* у (Hyvönen, и други, 2016).

Без подешавања параметара, *vp-tree* имплементира линеарну претрагу. Најлакши начин коришћења овог алгоритма на апроксимативан начин, подразумева ограничавање броја листова који се могу посетити. То се постиже подешавањем параметра *maxLeavesToVisit*.

Annoy алгоритам се ослања на *2-means-tree* структуру и векторске пројекције. У свакој итерацији при креирању стабла, хеуристиком се бирају два центроида који дефинишу хиперраван поделе. Креирањем више таквих стабала, то јест подешавањем параметра *n_trees*, повећава се прецизност алгоритма, али и величина индекса и време конструисања. Као и код *vp-tree,* постоји параметар за ограничавање броја листова који се обилазе у фази претраге *search_k.*

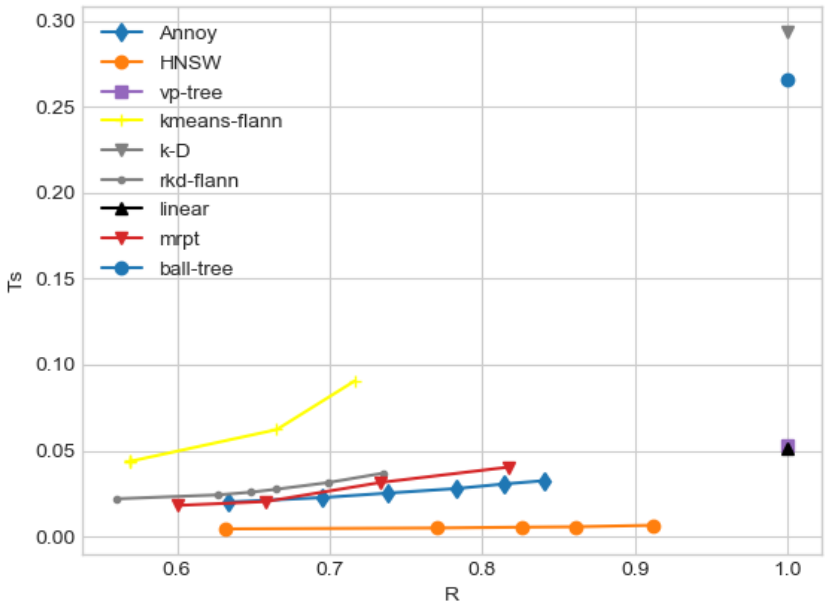
MRPT се заснива на *RP*-стаблима. На случајан начин се бира хиперраван за поделу, вектори се пројектују на раван, па се елементи смештају у лево и десно подстабло у односу на медијалну вредност пројекције. Могуће је креирање више оваквих стабала.

Испрва се резултати генеришу за свако стабло, а потом се из тако добијене листе кандидата селектују најближи. Ова библиотека нуди самоподешавајућу методу за коју је потребно дефинисати тражени одзив. Могуће је ограничавање броја стабала. Параметри су *target_recall* и *trees_max*.

Алгоритами у *flann* библиотеци названи су по алгоритмима којима су били инспирисани. Алгоритам *rkdTree* је заправо *randomized k-D tree*, док је *k-means* хијерархијско *k-means* стабло.

4.2 Селекција алгоритама

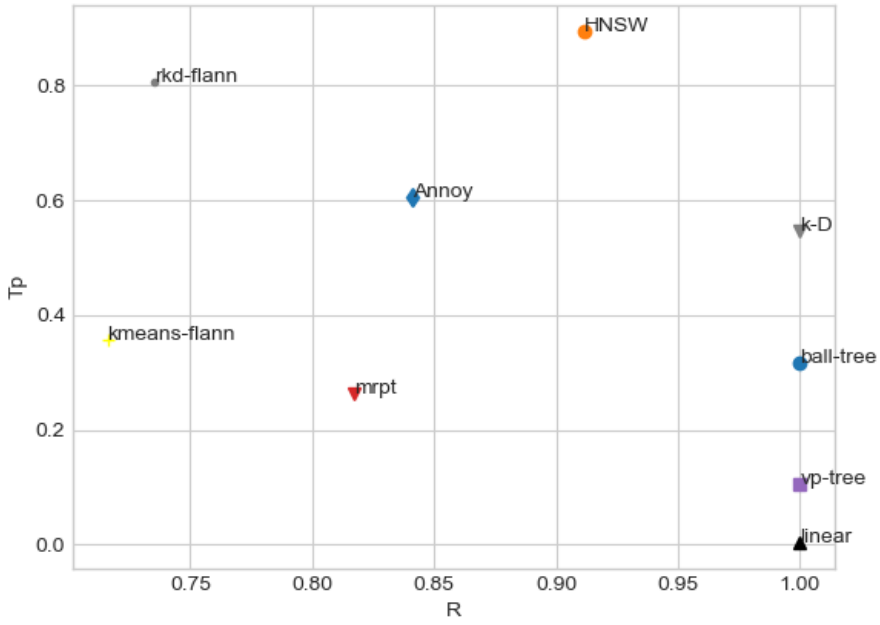
Како је за анализу већег броја података потребно више времена, пожељно је извршити тестирање алгоритама на мањем узорку *SIFTsmall*, са минималним подешавањем параметара. Први круг евалуације је започет на овом скупу података, како би се на брз начин проверили ефекти параметара. Мера удаљености на свим скуповима је еуклидско растојање. Поједине библиотеке нуде аутоматско прилагођавање параметра скупу података (*flann*, *MRPT*), док се друге ослањају на предефинисане вредности (*scikit-learn*) и ручно унете параметре (*Annoy*, *nmslib*).



Слика 6: Одзив и време претраге *SIFTSmall*

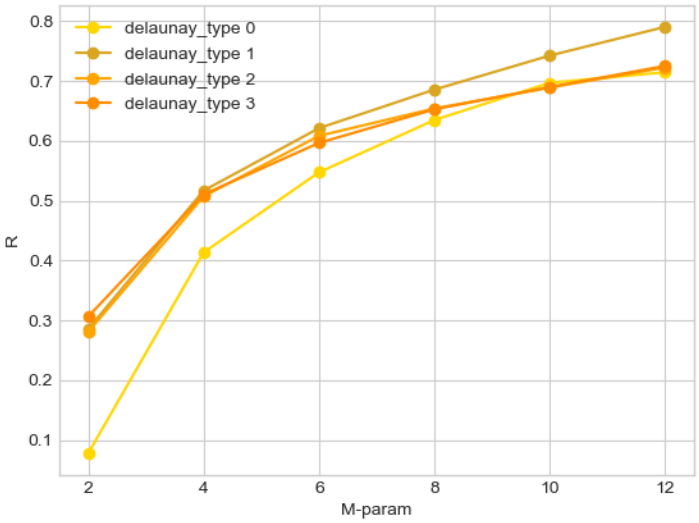
На малом узорку са већим бројем димензија, *k-D* показује слабије перформансе, што потврђује теоријске претпоставке. Предефинисани максимални број листова у стаблу је 100. Егзактне методе које се ослањају на структуру бинарних стабала биће искључене из детаљне анализе (*k-D* и *ball-tree*). За *Annoy* алгоритам је дефинисано 8 стабала, а

различита одзиви се постижу варирањем параметра *search_k*. Код конструкције *mrpt* алгоритма, унет је параметар за аутоматско подешавање *target_recall* и варира од 0.5 до 0.9, а максимални број стабала ограничен је на 10. У сладу са тим параметром, алгоритам је самостално дефинисао максималну дубину сатабала на 5 нивоа. Када је реч о алгоритмима *flann* библиотеке, код *kmeans* стабала, број грана (центроида) је подешн на 2, а варира број листова који се посећују у фази претраге. Код *rkd*, варира број стабала, а број лситова који се посећују је фиксан и износи 150. Пошто резултати нису оптимални, аутор саветује коришћење методе за самоподешавање код ове библиотеке. Најбржи алгоритам је *HNSW* (Слика 6). Параметар *M*, који се прослеђује за време креирања структуре, регулише максималан број грана по чвору, на свим нивоима графа осим на најнижем. Његова вредност је подешена на 3. Исти параметар утиче на број нивоа хијерархије, меморијске захтеве и време креирања структуре. *HNSW* има дуго време креирања (Слика 7), али за конструкцију индекса користи две процесорске нити. Прослеђен је параметар *ef* који одређује величину листе кандидата која се користи у фази претраге, и креће се између 40 и 150 .



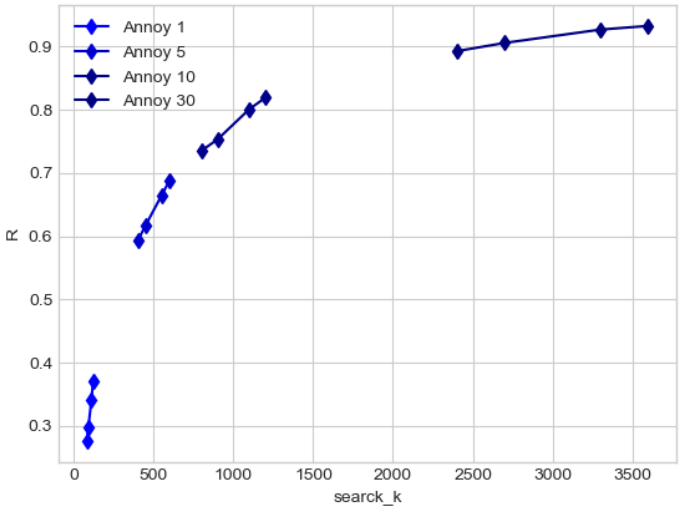
Слика 7: Одзив и конструкција индекса

Треба имати у виду да је аутоматско подешавање укључено у време конструисања *mrpt*. Пошто су *Annoy* и *HNSW* апроксимативне методе које постижу задовољавајући одзив за кратко време, над њима се врши анализа утицаја параметара.



Слика 8: *HNSW* - хеуристике

У теоријском опису *HNSW* је поменуто да постоји више хеуристика које врше претрагу, а користе се и за креирање графовске структуре. На тесту се показало да хеуристика 1 има бољи одзив када вредност параметра *M* расте, то јест када је број грана међу чворовима већи (Слика 8). Разлика између нулте хеуристике и осталих је већа, када је број грана у графу нижи. Када је број грана расте, не постоји значајна разлика међу хеуристикама 0, 2 и 3. Граф је боље повезан у том случају, па је претпоставка да је за добар одзив хеуристике 0 заслужна управо повезаност графа. Параметар *M* утиче на време креирања индекса и величину меморије коју они заузимају, те ако постоје меморијска ограничења на кластеру, за бољи одзив је пожељно користити хеуристике 1, 2 и 3. Треба имати у виду да оне утичу на време претраге, па је потребно извршити анализу, како би се постигао одговарајући одзив за жељено време. Параметар *ef* који утиче на време претраге, испитаће се у наредним поглављима.

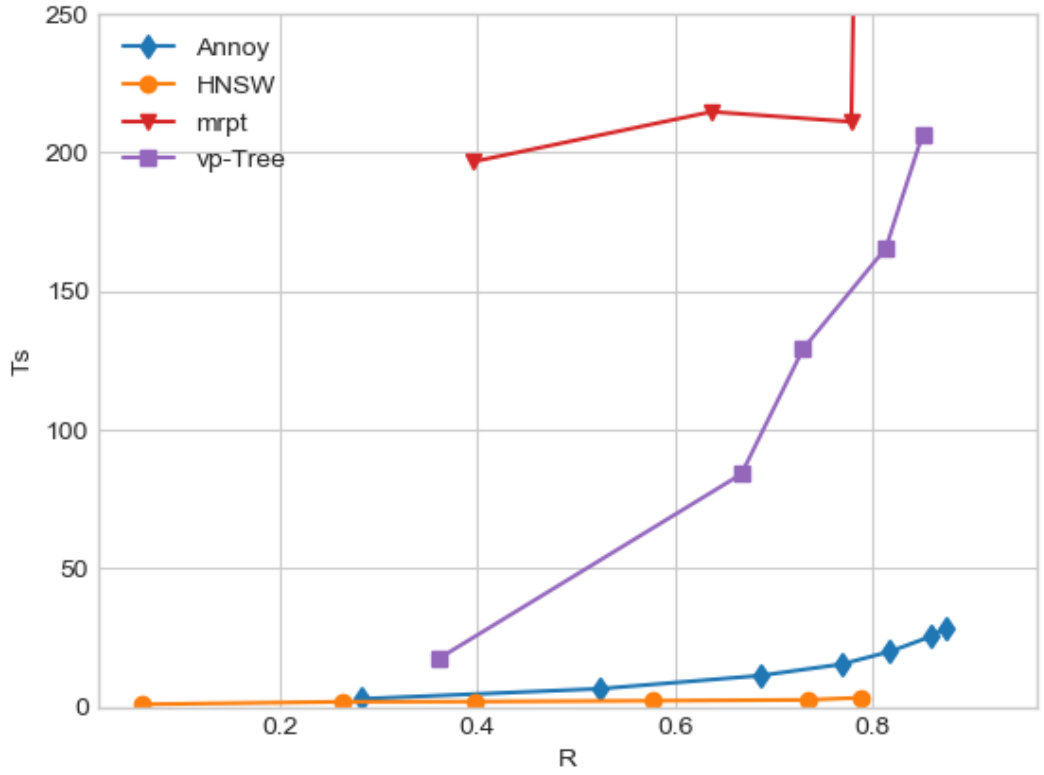


Слика 9: *Annoy*- утицај параметара

Код *Annoy* алгоритма је тестиран утицај параметара за претрагу *search_k*, при различитом броју конструисаних стабала. Предефинисана вредност параметра једнака броју стабала помноженом са бројем суседа који се траже (Слика 9). Овај параметар директно утиче на време претраге и подешава се након конструисања индекса. Примећује се да при већем броју стабала, параметар у мањој мери утиче на одзив алгоритма. Број стабала директно утиче на време конструисања индекса и меморију коју заузима, али има и пресудан утицај на време претраге и на одзив. Овде је потребно наћи прави однос параметара, како би се постигао одређени одзив у жељеном времену, што је случај и са осталим алгоритмима. Ако меморија представља ограничавајући фактор, па је број стабала фиксиран, повећање параметра *search_k* по цени споријег одзива је смислено. Ако је пак време критичан фактор, и потребан је виши одзив, може се креирати више стабла, али незнатно смањити вредност параметра *search_k.*

Почетни резултати на већем скупу података *SIFTM*, на персоналном рачунару, указују да алгоритми *Annoy* и *HNSW* дају резултате за значајно краћи временски период од других алгоритама (Слика 10). Време претраге *HNSW* мери се секундама, док се код осталих ради о десетинама и стотинама секунди. И поред подешавања параметра *M*, алгоритам није забележио задовољавајући одзив. Најбољи одзив постигнут је са параметром *M*=40, а препоручене вредности параметра су између 2 и 48. Веома добре резултате постиже *Annoy,* где је најбољи одзив од 0.85 постигнут са 80 стабала. Одзив од 0.81 постигнут је са 60 стабала. Иако је спор, изненађујуће резултате постиже *vp-tree* коме је максималан број елемената у листу подешен на десет хиљада, што је 1% тест скупа података. Најбољи одзив од 0.85 постиже се када је параметар *maxLeavesToVisit*

25 – што не може бити више од 25% скупа података. Алгоритам *mrpt* подешен је преко методе за аутоматско формирање параметара и одзив од 0.77 постиже са 262 стабла. Максималан број стабала који је прослеђен као параметар претраге је 300. Са 458 стабала, постиже се одзив од 0.92 али за изузетно дуг временски период, па је овај алгоритам изузет из даље анализе.



Слика 10: *SIFT1M* - почетни резултати

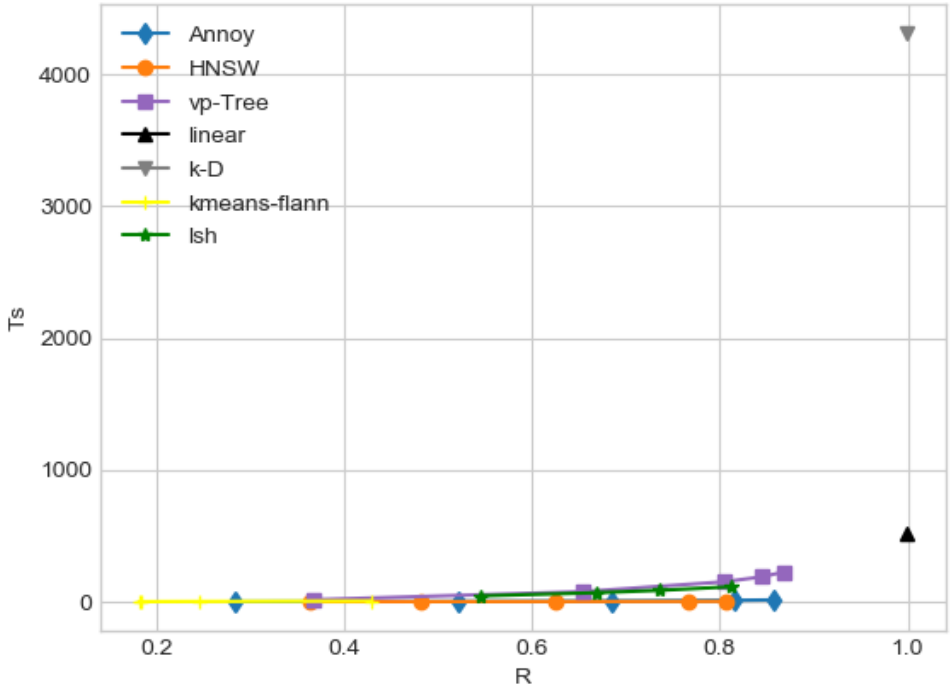
Даље истраживање се спроводи на удаљеном рачунару, под *Linux* оперативним системом, па је могуће тестирати и *LSH* алгоритам из *FALCONN* библиотеке. Ради поређења резултата, извршена су и линеарна претрага и *k-D* алгоритам. Избор параметара код *flann* библиотеке, препуштен је функцији за аутоматско тражење параметара, али резултати алгоритма нису задовољавајући. Наведена функција бира параметре претраге и алгоритам претраге на основу узорка из скупа података и на основу коефицијената који регулишу време креирања структуре и меморијске захтеве (*build_weight* ,*memory_weight*).

Параметри метода дати су у табели:

Алгоритам	Параметри
Annoy	Број стабала *n_trees* = 5, 15, 20, 60, 80
HNSW	Број грана *M* = 5, 8, 15, 30, 38
vp-Tree	Величина листа *bucketSize* = 10000; *maxLeavesToVisit* = 2, 10, 20, 25, 30
k-D	Величина листа *leafSize* = 10000
flann	*target_precision* = 0.3, 0.6, 0.7, 0.85 *build_weight* = 0; *memory_weight* = 0,*sample_fraction* = 0.05
LSH	Број табела *t*= 15; Број хеш функција *k* = 4; *n_ probes* = 15, 30, 45, 75

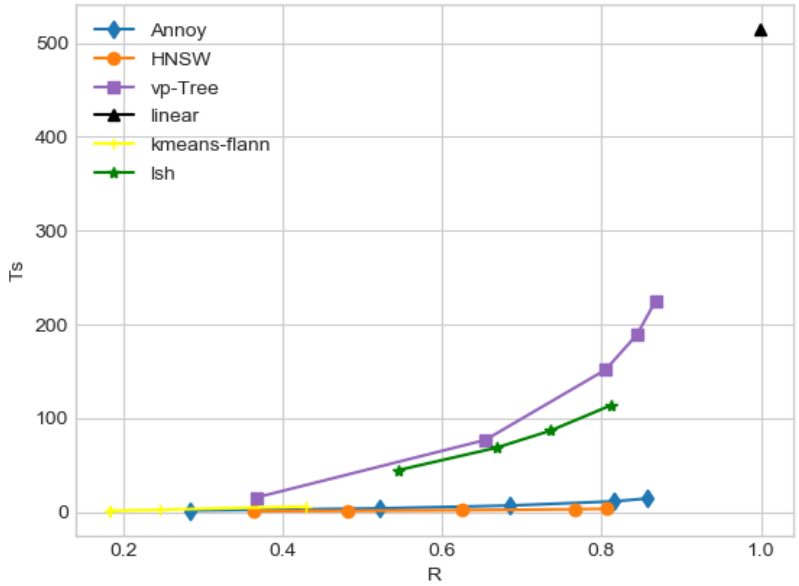
Табела 3: Параметри на *SIFT1M* *DataBricks*

Све апроксимативне методе, осим оних из *flann* библиотеке, постигле су одзив већи од 0.8 (Слика 11). Најгоре време претраге имао је *k-D* и чије време извршавања износи више од 4300 секунди. То је скоро 9 пута спорије од линеарне претраге за коју је задужена *scikit-learn* библиотека.



Слика 11: Одзив и Ts над *SIFT1M* на DataBricks

Детаљни приказ добијених резултата се налази у Прилог 1, а делимичан је дат у Табела 4. Због јаснијег прегледа је искључен *k-D* приступ (Слика 12).



Слика 12: Време претраге и одзив на *SIFT1M*

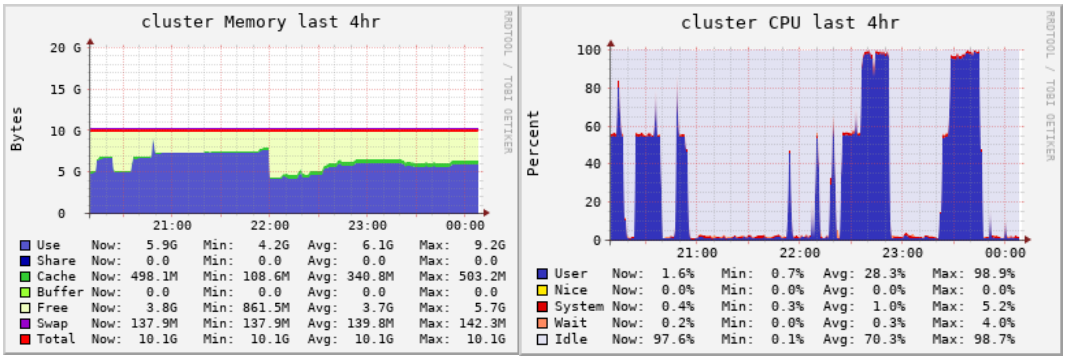
Линеарна претрага над овим скупом података је релативно брза и завршена је за 515 секунди (Табела 4). Апроксимативни алгоритми су остварили одзив већи од 0.8 за време које је и до 100 пута мање у односу на линеарну претрагу (*HNSW*). Просечна удаљеност тачака претраге од идентификованих суседа је незнатно већа од стварне, која је одређена потпуном претрагом скупа података (*linear*). Подаци о удаљености нису доступни за *LSH*.

Алгоритам	T_p	T_s	R	Просечна удаљеност	E
Annoy-trees-80	244.18	14.5923	0.8574	236.89	100.53%
HNSW-M-38	3269.57	3.627	0.8064	237.35	100.73%
vp-Tree-10k-mL30	17.5	225.437	0.8681	237.83	100.93%
linear	0.09	515.1839	1.0	235.63	100%
Lsh t15 k4 t75	132.35	114.1396	0.8125	0.0	/

Табела 4: *SIFT1M* на *DataBricks* - издвојени резултати

Приметна је велика разлика у времену потребном за конструисање индекса. *HNSW* је утрошио највише времена и ресурса за изградњу структуре и томе је највише допринео сложен начин формирања индекса и већи меморијски захтеви због броја грана у графу (M=48). У појединим интервалима, процес креирања је у потпуности заузео ресурсе процесора (Слика 13), што није био случај код свих алгоритама. Резултати показују да

HNSW, *Annoy* и *LSH* имају боље перформансе у односу на остале алгоритме. Они су уједно и представници група које су описане у другом поглављу.



Слика 13: Меморијски (лево) и рачунски захтеви (десно)

Сва три алгоритма захтевају већи меморијски простор и пружају могућност складиштења индекса на диску рачунара. После једног од експеримената над *SIFT1M* подацима на персоналном рачунару, сачувани индекси су заузели више меморије од самог скупа података! *Annoy* од 80 стабала заузео је 1.4 *Gb*, а *HNSW* 0.9 *Gb* меморије.

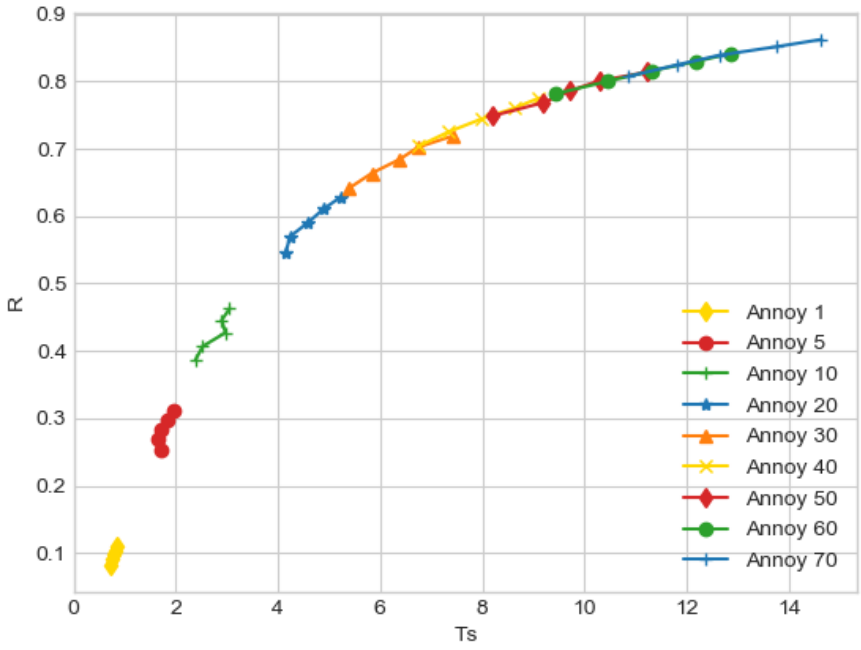
Иако је скуп података *SIFTSmall* био од помоћи када је требало стећи увид у параметре *Annoy*, *HNSW* и других алгоритма, ипак није довољано велики да би се направила значајна дистинкција међу њима. Јаснији преглед се стиче над *SIFT1M*, где се примећују веће разлике у перформансама између линеарне претраге и апроксимативних алгоритама. Разлике у времену претраге су учљиве и међу самим припадницима апроксимативних метода *mrpt*, *vp-Tree*, *LSH*, *Annoy* и *HNSW* при вишим одзивима. Аутор није успео да побољша перформансе алгоритама из *flann* библиотеке мануелним комбиновањем параметара, нити аутоматским подешавањем у последњем експерименту. Алгоритми су најчешће тестирани варирањем параметара који утичу на формирање структуре попут броја грана графа, стабала, хеш табела и величине листова, док ће се у наредним поглављима над одабраним приказати комбиновани утицај параметара конструкције и претраге. Из изложеног се зајључује да *HNSW*, *Annoy* и *LSH* имају висок одзив и ниже време претраге од осталих алгоритама, па се ефекти њихових параметара детаљније анализирају.

4.3 Анализа параметара

Како је наведено, у наредним поглављима детаљније се приказује утицај параметара над *SIFT1M* подацима код *Annoy*, *LHS* и *HNSW*.

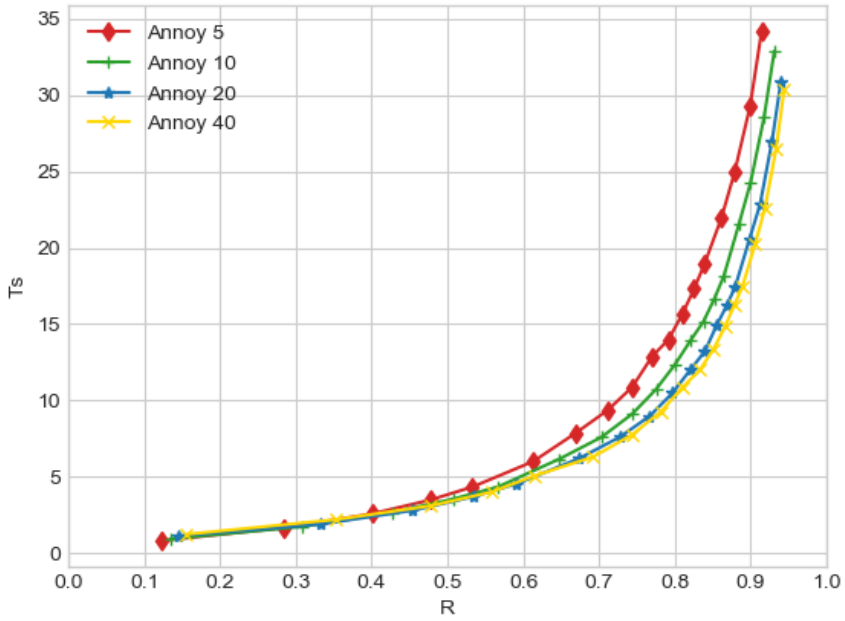
4.3.1 *Annoy*

На Слика 14 је приказан однос времена претраге и одзива, у зависности од броја стабала и параметра *search_k* који, као што је наглашено, утиче на T_s. Утицај броја стабала на однос индикатора перформанси је занемарив у појединим случајевима. Разлика у одзиву између алгоритма са 60 и 70 стабала у дванаестој секунди је минорна, па се намеће питање колико је стабала потребно да би се постигао жељени однос одзива и времена претраге. Већи број стабала заузима више меморије (Прилог 2), па је оптимизација оправдана када су меморијски капацитети ограничени.



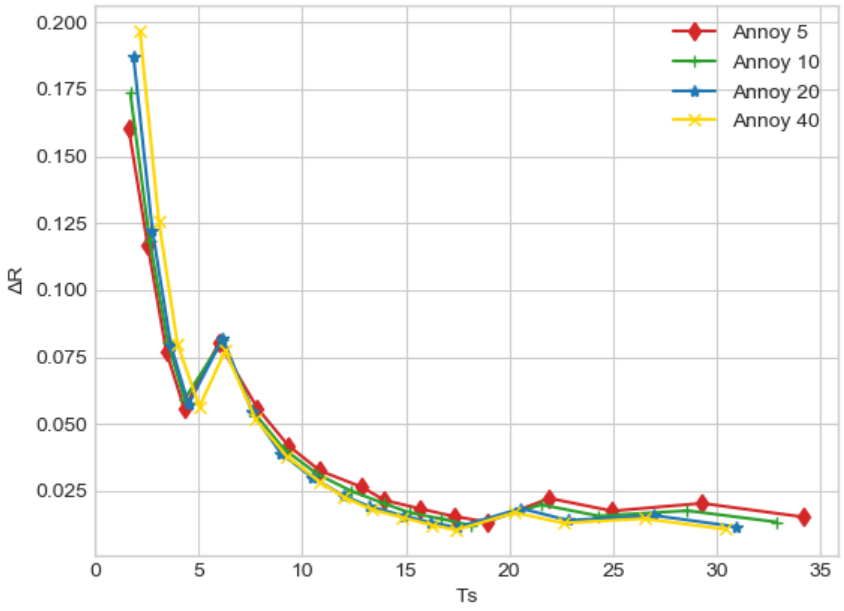
Слика 14: *Annoy* - Број стабала и перформансе

Може се претпоставити да је у наведеном случају коришћено много више стабала него што је заиста потребно, па сличан бој стабала има готово истоветне резултате. Наведену претпоставку могуће је проверити искуственим путем на мањем броју стабала (Слика 15). При нижем одзиву, све варијанте алгоритма са различитим бројем стабала, постижу сличне перформансе, а исти ефекат јавља се и у претходно описаном случају. Са порастом одзива долази до раслојавања и алгоритми са мањим бројем стабла имају дуже време претраге – одзив од 0.9 је постигнут за 20 секунди код експеримента са 40 стабала, што је скоро 10 секунди брже од експеримента са 5 стабала.



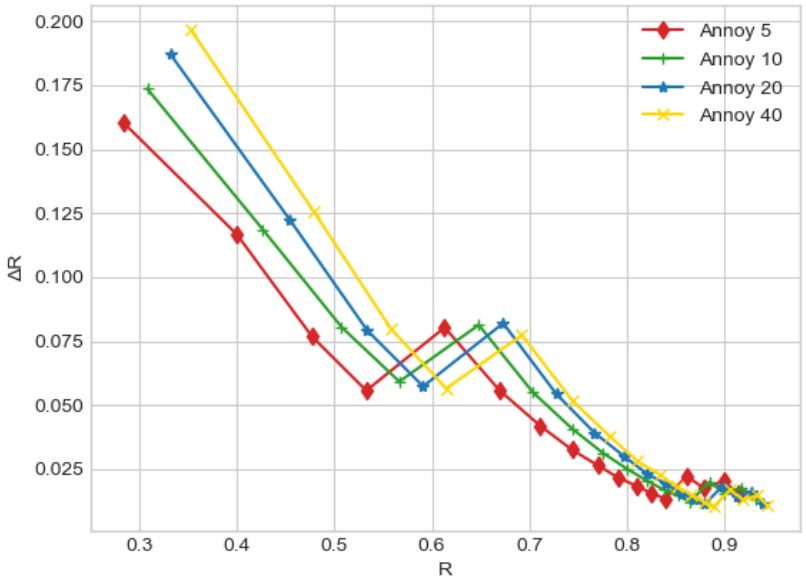
Слика 15: *Annoy*- раслојавање

Маргинални одзив се смањује са дужином претраге (Слика 16). Дијаграм показује да је маргинални одзив незнатно нижи код већег броја стабала при дужем времену претраге.



Слика 16: *Annoy*- маргинални одзив и време претраге

Када се маргинални одзив упореди са постигнутим одзивом (Слика 17), јасно је да су алгоритми са већим бројем стабала у предности у погледу одзива.

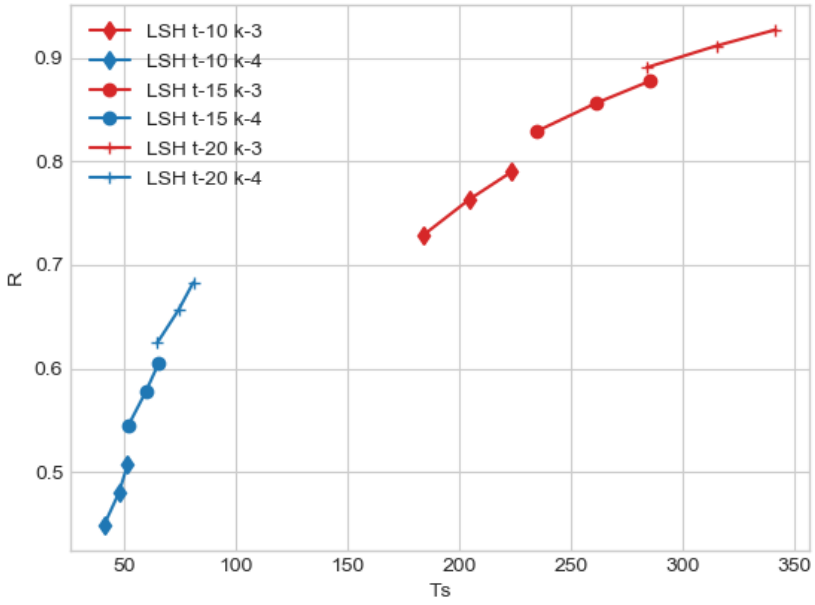


Слика 17: *Annoy* - маргинални одзив и одзив

Скокови на дијаграму су последица методолошког поступка. Параметар *search_k je* у прве четири итерације увећаван за 500, а у следећим итерацијама се увећавао за 1000 због бржег добијања резултата. На основу приказаних резултата, закључује се да је могуће оптимизовати број стабала алгоритма, што је веома важно када су присутна строга меморијска ограничења, то јест смањити број стабала а повећати параметар претраге.

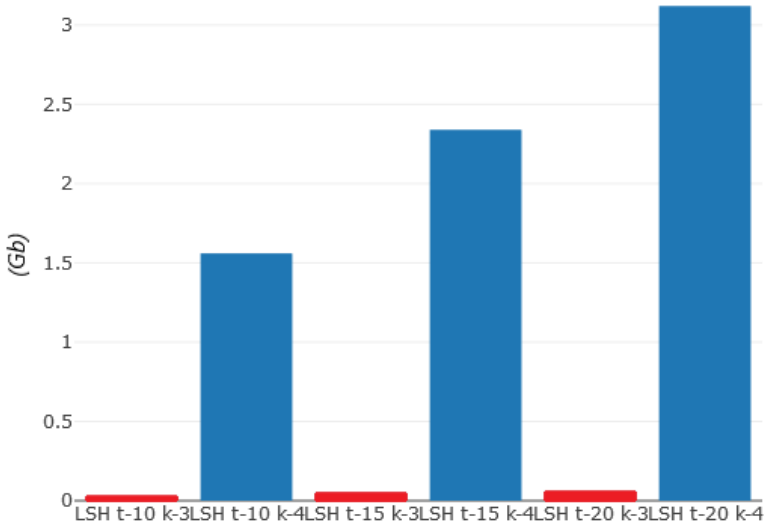
4.3.2 LSH

Утицај броја табела *t* и хеш функција *k* код *LSH* приказан је на Слика 18. Резултати се групишу у односу на величину параметра *k*. Плавом бојом означене су варијанте у којима је $k = 4$, а црвеном када је $k = 3$. Из приказаног се закључује да параметар има висок утицај на одзив, јер нижи број хеш функција производи мањи број корпи - које постају гломазне. Величина корпе утиче на раст времена претраге, али је чини прецизнијом јер се испитује велики број кандидата. Ипак, време претраге од 200 секунди је неприхватљиво када имамо у виду резултате других алгоритама, па би анализу требало наставити са параметром $k = 4$. Нижи одзив који је карактеристичан за за ту групу се може побољшати повећањем броја „суседних“ корпи који се претражују.



Слика 18: *LSH* - утицај параметара на перформансе

Параметар *k* не утиче само на одзив и време претраге, већ и на велике меморијске захтеве, у комбинацији са бројем хеш табела (Слика 19). Показљељи заузетости меморије су значајно већи него код осталих алгоритама и постигнути су за релативно кратко време конструисања (Прилог 3).

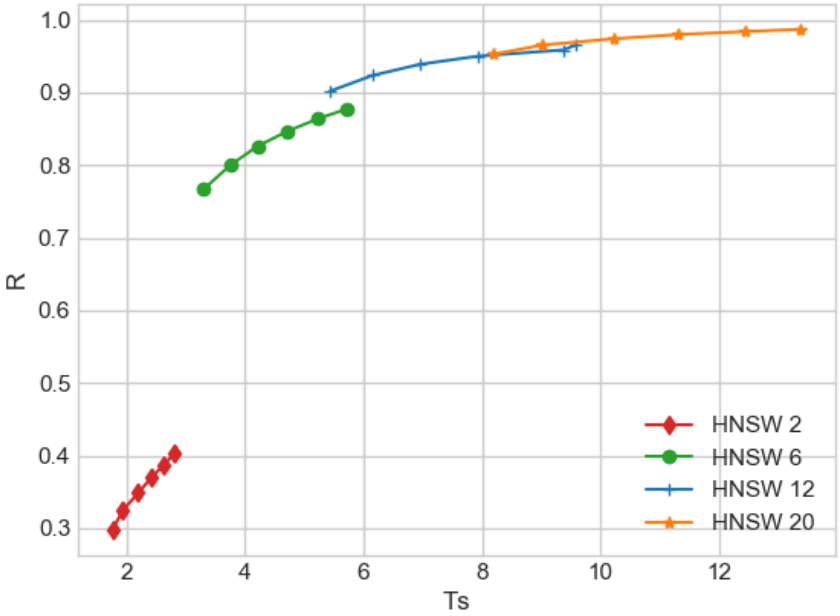


Слика 19: *LSH*- меморија

Резултати тестирања *LSH* указују да има ниже перформансе у поређењу са другим алгоритмима јер генерише мањи одзив за више времена и користећи више меорије од *Annoy* и *HNSW*.

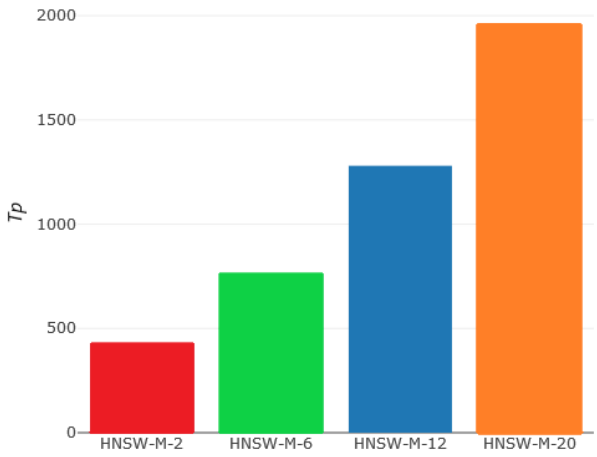
4.3.3 HNSW

HNSW је на тестовима показао најбоље перформансе и једини је алгоритам који достиже висок одзив за веома кратко време претраге (Слика 20). Одзив од 0.98 је постигнут у року од 13 секунди. Подсећања ради, линеарна претрага је генерисала резултат након више од 515 секунди.

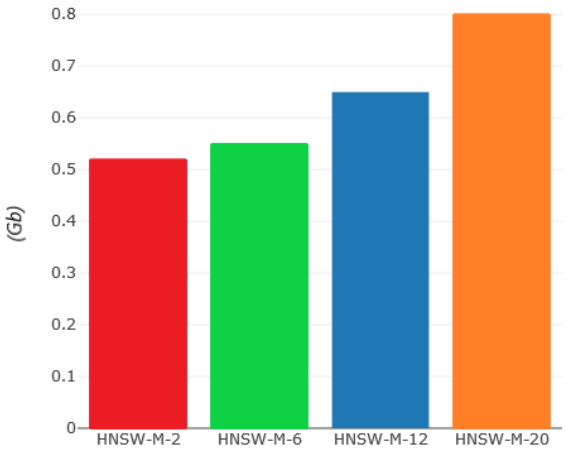


Слика 20: *HNSW* - одзив и време претраге

Можемо закључити да је и код овог алгоритма, као и код *Annoy*, могуће постићи сличне резултате различитим комбиновањем параметара за изградњу структуре M и параметром за подешавање броја чворова који се посећују у фази претраге ef. Оно по чему се *HNSW* издваја је дуго време конструисања индекса (Слика 21) и изненађујуће, нижим меморијским захтевима у поређењу са претходним алгоритмима (Слика 22).



Слика 21: *HNSW- Tp*



Слика 22: *HNSW*- меморија

Верзија *HNSW* са *M=12* има упоредиве меморијске захтеве као *Annoy* од 60 стабала, али генерише одзив већи од 0.9 за мање од 6 секунди, док *Annoy* постиже одзив од 0.8 у десетој секунди. Овај податак сврстава *HNSW* на прво место по перформансама у досадашњем тестирању.

4.4 Анализа утицаја на предвиђање

Одабрани алгоритми се тестирају на релативно малом, али високо димензионираном *FashionMINST* скупу података, где је потребно класификовати фотографије у десет одевних категорија (класа), као што је напоменуто у поглављу 4.1. Како је наглашено, над овим подацима је могуће тестирати квалитет предикције, па су у складу са тим

поменуте две мере квалитета предвиђања, макро прецизност (P_m) и макро одзив (R_m), које се дефинишу:

$$P_m = \frac{1}{c}\sum_{i=1}^{c}\frac{TP_i}{TP_i + FP_i}$$

$$R_m = \frac{1}{c}\sum_{i=1}^{c}\frac{TP_i}{TP_i + FN_i}$$

где је

c - број категорија

TP_i – број правилно класификованих инстанци класе i,

FP_i – број припадника других класа, који су погрешно сврстани у класу i

FN_i – број припадника класе i, који су погрешно сврстани у друге класе.

Квалитет предвиђања је бољи када вредност наведених индикатора тежи јединици, што је уједно и њихова горња граница.

Резултати тестирања и услови под којима су добијени, приказани су у Табела 5:

Алгоритам	T_p	T_s	R	Параметри	P_m	R_m	E
linear	/	71.90	1	/	0.82330	0.81680	100%
Annoy-trees-10	10.83	8.04	0.80944	*n_trees* = 10 *search_k* = 1000	0.81031	0.80599	100.65%
Annoy-trees-10	10.83	10.74	0.88895	*n_trees* = 10 *search_k* = 2000	0.81624	0.8116	100.41%
Annoy-trees-10	10.83	13.00	0.92432	*n_trees* = 10 *search_k* = 3000	0.81784	0.8129	100.28%
Annoy-trees-10	10.83	15.53	0.94514	*n_trees* = 10 *search_k* = 4000	0.81786	0.8123	100.20%
HNSW-M-4	51.76	3.81	0.91525	*M*=4 *ef*=100	0.81669	0.809	100.36%
HNSW-M-4	51.76	4.96	0.95001	*M*=4 *ef*=140	0.81839	0.81099	100.19%
HNSW-M-4	51.76	6.40	0.97124	*M*=4 *ef*=200	0.82040	0.8133	100.11%
lsh t-10 k-3 p-10	37.18	51.94	0.63694	*t*=10; *k*=2; *n_probes* = 10	0.78257	0.7825	/
lsh t-10 k-3 p-14	37.18	64.00	0.67879	*t*=10; *k*=2; *n_probes* = 14	0.79029	0.7881	/
lsh t-10 k-3 p-20	37.18	71.28	0.71730	*t*=10; *k*=2; *n_probes* = 20	0.79542	0.7913	/

Табела 5: *FashionMINST* - резултати

Линеарна претрага је завршена за више од седамдесет секунди - што је брз резултат. Када би се предвиђање спровело на основу линеарне претраге, макро прецизност би износила 0.82. Занимљиво је да чак и најлошија апроксимација алгоритмом *Annoy*, има макро прецизност блиску оној, која је постигнута линеарном претрагом P_m= 0.81. Као и на претходним скуповима података и овде најбоље резултате има *HNSW* који у најгорем случају има макро одзив од 0.81. Најлошије резултате има *LSH*, где су се при тестирању користили различити параметри везани за број табела и хеш функција, али су у табели приказани најбољи. Метода за аутоматско одређивање параметра је препоручила вредности наведене у табели.

Из приказаног се закључује да су резлике у предвиђању апроксимативних алгоритама у односу на линеарну претрагу минимални, али доносе веће временске уштеде – код *HNSW* се упоредив резултат $P_m = 0.82$ добија десет пута брже ($T_s = 6.4$) , у односу на линеарну претрагу ($T_s = 71.9$).

4.5 Дискусија и препоруке

Од издвојених алгоритама, најбоље перформансе показује *HNSW,* јер пружа висок одзив за кратко време претраге. Мана алгоритма је дуг период креирања структуре и што није у потпуности јасно како се структура дистрибуира. Један од једноставних начина би подразумевао поделу података у чворовима дистрибуираног система, премда аутори тврде да постоје боља решења. Изненађујуће, алгоритам је користио мање меморије за изградњу структуре од *Annoy* и *LSH*. Најновија истраживања поменута у другом поглављу и (Baranchuk, Babenko, & Malkov, 2018), комбинују овај приступ са квантизацијом, како би прилагодили меморијске захтеве за податке од преко милијарду инстанци и задржали висок одзив и ниско време претраге. Други алгоритам који генерише добре резултате и заснива се на структури која је погодна за дистрибуирање је *Annoy*. Овај алгоритам је у употреби у компанији за медијске услуге *Spotify*, која располаже базама које имају више милиона корисника и песама. Аутору овог рада није познато како се алгоритам понаша над скуповима који су реда величине милијарде. Имплементација *LSH* у библиотеци *FALCONN* је имала лошије резултате од претходих алгоритама у погледу одзива, времена претраге и меморијских захтева. Пошто је то једини тестирани приступ који се заснива на хеширању, правци будућег истраживања би могли бити усмерени на методе који користе другачије хеш функције. Свакао би

требало тражити *Multi-Probe* имплементације, које омогућавају смањење меморијских захтева. На изненађење аутора, хијерархијско *kmeans* стабло *flann* библиотеке није показало очекиване перформансе због потешкоћа у подешавању параметара, па би будуће истраживање могло бити усмерено на свеобухватну анализу истих.

Када је реч о једноставности подешавања параметара алгоритама, *Annoy* се сврстава у најједноставније, јер захтева подешавање свега два параметра. Утицај параметара на кључне показатеље перформанси је лако утврдити, па та чињеница олакшава коришћење. Мали број параметара може представљати и ограничавајући фактор, јер се не може управљати процесом изградње структуре. Подешавање параметара код *HNSW* алгоритма није једноставно као у претходном случају, али пружа велику флексибилност. У раду су коришћена два параметра чију је улгоу лако објаснити. У референтној литератури може се наћи још параметара и анализа њиховог утицаја на перформансе. Аутор је имао највише потешкоћа са подешавањем свега три параметра *LSH* алгоритма, најпре због велике разлике коју прави број хеш функција које се користе.

Пре одабира одговарајућег алгоритма, аутор саветује идентификовање ограничења – меморије, времена претраге, времена конструисања структуре, дистрибуцију података и постојање кластера. У складу са ограничењима, пожељно је извршити тестове над подскупом података. Скалабилност алгоритама се може проверити постепеним повећавањем подскупа за тестирање. Пре коришћења над целим скупом је потребно изменити параметре како би се прилагодили новој количини података. У складу са тим је препоручена провера референтне литературе, која често садржи таква поређења. Како је показано у преходним поглављима, често постоји могућност за оптимизацију алгоритама, где се сличан одзив и време претраге постижу са различитим величинама структуре. Оптимизација се врши у односу на меморијска ограничења, жељени одзив и време претраге.

5. Закључак

Три алгоритма за апроксимативну претрагу суседа, који су показали најбоље перформансе, су детаљно описани у раду. Пружена је анализа њихових најважнијих параметара на кључне индикаторе перформанси, што олакшава одабир алгоритама и њихова подешавања. Потврдило се да апрокисмативне методе могу значајно смањити време претраге, без пресудног утицаја на њен квалитет и предвиђање. Приказана је теоријска основа за одабране алгоритме и предложене су смернице за њихову употребу. Дат је разноврстан приказ литературе из области апроксимативне претраге суседа, који заинтересованим појединцима може користити за будућа истраживања. У свету растућих података, апроксимативна претрага суседа има и имаће пресудну улогу.

6. Референце

Abbasifard, M. R., Ghahremani, B., & Naderi, H. (2014). A survey on nearest neighbor search methods. *International Journal of Computer Applications, 95*, 39-52.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., & Schmidt, L. (2015). Practical and optimal LSH for angular distance. *Advances in neural information processing systems*, (стр. 1225-1233).

Arya, S., Mount, D. M., & Narayan, O. (1996). Accounting for boundary effects in nearest-neighbor searching. *Discrete & Computational Geometry, 16*, стр. 155-176.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM), 45*, 891-923.

Baranchuk, D., Babenko, A., & Malkov, Y. (2018). Revisiting the inverted indices for billion-scale approximate nearest neighbors. *Proceedings of the European Conference on Computer Vision (ECCV)*, (стр. 202-216).

Boytsov, L., & Bilegsaikhan, N. (2013). Engineering Efficient and Effective Non-metric Space Library. *Similarity Search and Applications - 6th International Conference* (стр. 280-293). Springer.

Douze, M., Sablayrolles, A., & Jégou, H. (2018). Link and code: Fast indexing with graphs and compact regression codes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (стр. 3646-3654).

Douze, M., Sablayrolles, A., & Jégou, H. (2018). Link and code: Fast indexing with graphs and compact regression codes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (стр. 3646-3654).

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *Vldb, 99*, стр. 518-529. Edinburgh.

Han, J., Pei, J., & Kamber, M. (2012). *Data mining: concepts and techniques* (3rd изд.). Waltham, Massachusetts, United States of America: Elsevier.

Hyvönen, V., Tasoulis, S., Jääsaari, E., Tuomainen, R., Wang, L., Corander, J., & Roos, T. (2016). Fast nearest neighbor search through sparse random projections and voting. *2016 IEEE International Conference on Big Data (Big Data)*, (стр. 881-888).

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. ACM.

Jegou, H., Douze, M., & Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, стр. 117-128.

Kleinberg, J. M. (1997). Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (стр. 599--608). El Paso,Texas, USA: ACM.

Kushilevitz, E., Ostrovsky, R., & Rabani, Y. (2000). Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing, 30*, 457-474.

Liu, T., Moore, A. W., Yang, K., & Gray, A. G. (2005). An investigation of practical approximate nearest neighbor algorithms., (стр. 825-832).

Maillo, J., Ramirez, S., Triguero, I., & Herrera, F. (2017). kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems, 117*, 3-15.

Malkov, Y., A, & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*.

Malkov, Y., Ponomarenko, A., Logvinov, A., & Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, стр. 61-68.

Marimont, R., & Shapiro, M. (1979). Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, стр. 59-70.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 331-340.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Prettenhofer, P. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, стр. 2825-2830.

Rodola, G. (2008). *github*. Преузето са github: https://github.com/giampaolo/psutil

Samet, H. (2006). *Foundations of multidimensional and metric data structures.* (J. Gray, Ур.) San Francisco, California, United States of America: Morgan Kaufmann.

Seidl, T., & Kriegel, H.-P. (1998). Optimal Multi-Step k-Nearest Neighbor Search. *27*. Seattle: ACM SIGMOD Record.

Spotify Technology S.A. (2015). *Annoy*. Преузето са github: https://github.com/spotify/annoy

Triguero, I., Maillo, J., Luengo, J., García, S., & Herrera, F. (2016). From big data to smart data with the k-nearest neighbours algorithm.

UCI Machine Learning Repository. (n.d.). Преузето са University of California, Irvine: https://archive.ics.uci.edu/ml/datasets.html

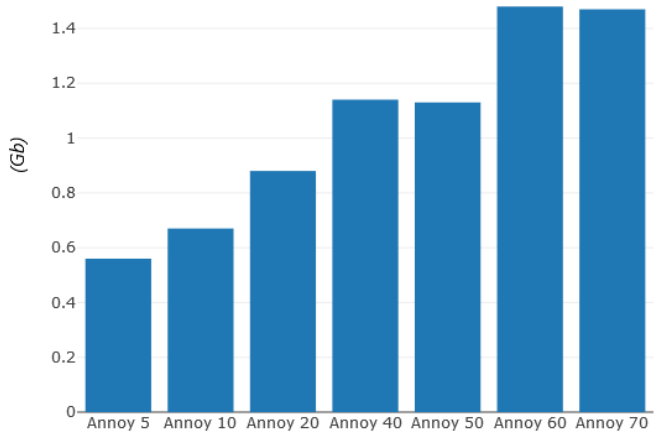
Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *e 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, (стр. 311-321). Austin, USA.

7. Прилози

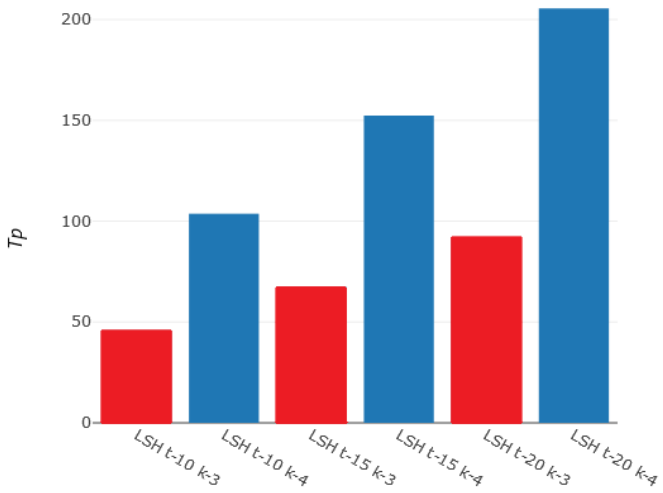
Прилог 1:Почени резултати на *SIFT1M* подацима

Алгоритам	Tp	Ts	R	Просечна удаљеност
Annoy-trees-5	35.93	1.7853	0.2835	254.7
Annoy-trees-15	62.39	4.0106	0.5221	243.33
Annoy-trees-30	102.71	7.0456	0.6841	239.47
Annoy-trees-60	185.83	11.5536	0.8158	237.4
Annoy-trees-80	244.18	14.5923	0.8574	236.89
HNSW-M-5	633.89	1.1619	0.3629	251.36
HNSW-M-8	856.31	1.4383	0.4822	244.77
HNSW-M-15	1347.66	2.0843	0.6244	240.4
HNSW-M-30	2519.09	3.1523	0.7668	237.86
HNSW-M-38	3269.57	3.627	0.8064	237.35
vp-Tree-10k-mL2	17.5	15.7579	0.3673	252.0
vp-Tree-10k-mL10	17.5	76.9463	0.6541	241.96
vp-Tree-10k-mL20	17.5	151.8395	0.8051	238.83
vp-Tree-10k-mL25	17.5	189.0328	0.8444	238.17
vp-Tree-10k-mL30	17.5	225.437	0.8681	237.83
linear	0.09	515.1839	0.9999	235.63
k-D	30.08	4317.1864	0.9999	235.63
autotuned-flann-build005	463.1	1.4064	0.1831	277.67
autotuned-flann-build005	445.64	1.3903	0.1811	278.17
autotuned-flann-build005	454.99	2.6843	0.246	260.7
autotuned-flann-build005	753.91	6.0138	0.4294	246.94
lsh-l15k4t15	132.35	44.8974	0.5454	0.0
lsh-l15k4t30	132.35	68.6406	0.6679	0.0
lsh-l15k4t45	132.35	86.9347	0.7362	0.0
lsh-l15k4t75	132.35	114.1396	0.8125	0.0

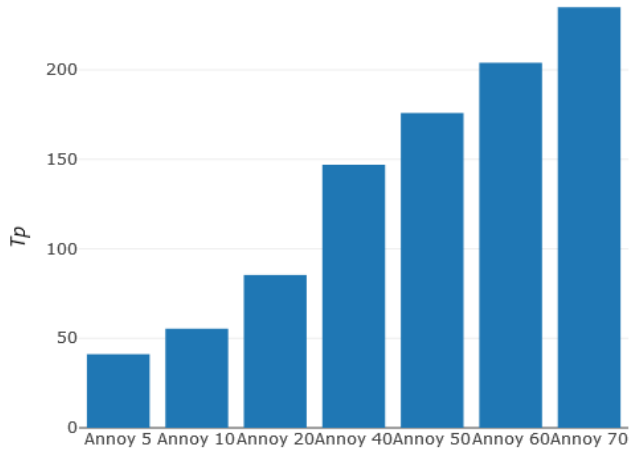
Прилог 2: *Annoy* - меморијски захтеви



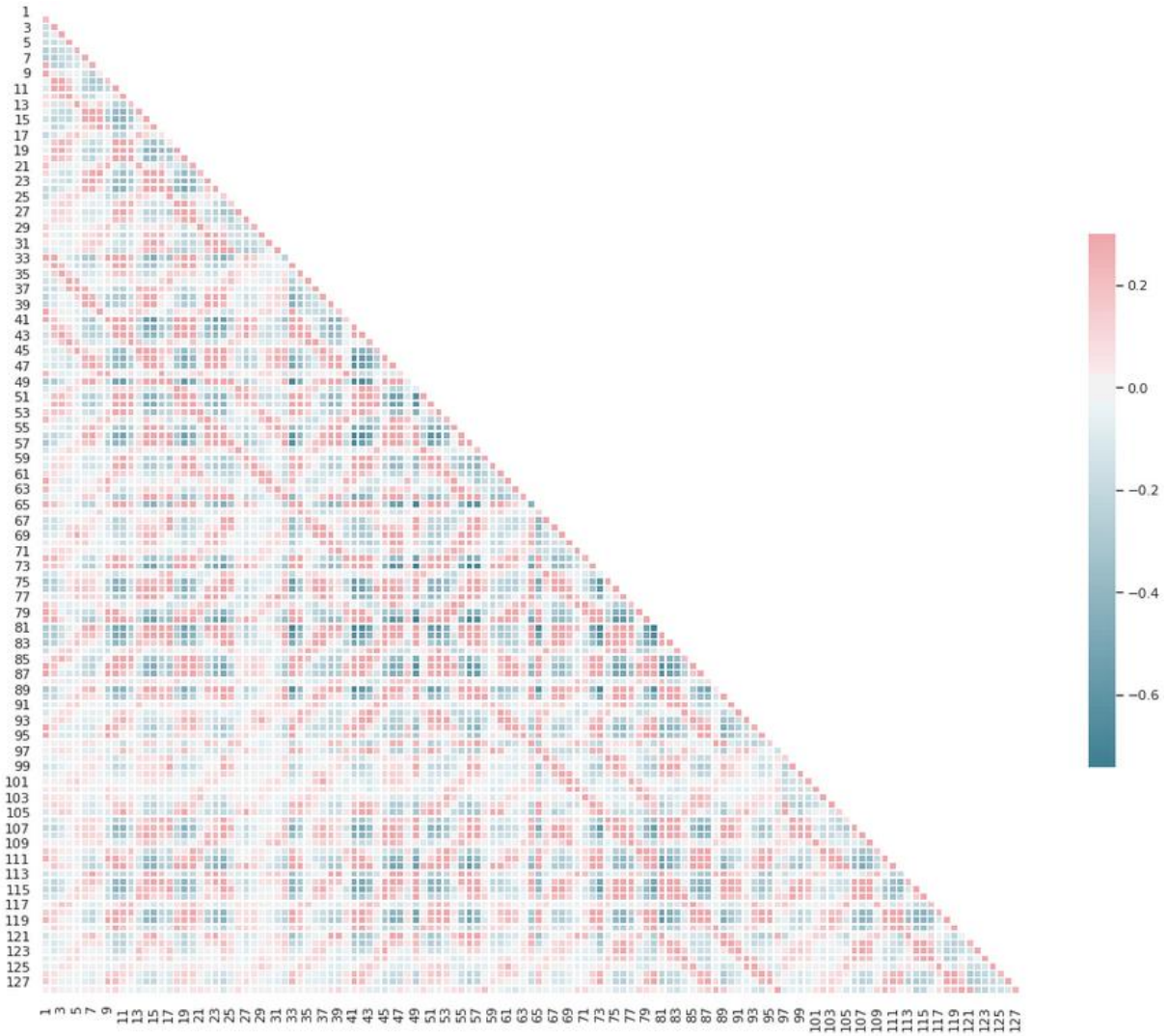
Прилог 3: *LSH* - Tp



Прилог 4: *Annoy* - Tp



Прилог 5:*SIFT1M* - Корелација



Прилог 6: *Annoy* - код за анализу параметара

```
annoyResults = []

numTreesParam = []

searchKparam = []

numTreesParam = []

from annoy import AnnoyIndex

for trees in [2,10,20,30,40,50,60,70]:

    numTrees = trees


    f = train.shape[1]

    t = AnnoyIndex(f, 'euclidean')
```

```
startMemory = getUsedMemory()
startClock= time.clock()
startTime = process_time()
for i in range(train.shape[0]):
    t.add_item(i,train[i])
t.build(numTrees)
end_time = process_time()
constructionTime = end_time - startTime
endClock = time.clock()
constructionClock= endClock - startClock
endMemory = getUsedMemory()
memoryConsumption = endMemory -  startMemory

for search in [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000]:
    searchK = search
    numTreesParam.append(numTrees)
    searchKparam.append(searchK)
    rez = []
    dist = []
    startClock = time.clock()
    startTime = process_time()
    for q in query:
        res,d = t.get_nns_by_vector(q, 100, search_k = searchK, include_distances=True)
        rez.append(res)
        dist.append(d)
    end_time = process_time()
    searchTime = end_time - startTime
    endClock = time.clock()
```

```
    searchClock= endClock - startClock
    result = fillIfNotAllAreFound(rez)
    annoyResults.append(result)
    result = np.asanyarray(result)
    annoyRecall = returnRecall(result, groundTruth)
    avgDist = np.mean(list(chain.from_iterable(dist)))
    reacll.append(annoyRecall)
    algorithm.append('Annoy-trees-'+str(numTrees))
    construciotnTimes.append(constructionTime)
    searchTimes.append(searchTime)
    avgdistances.append(avgDist)
    searchClocks.append(searchClock)
    constructionClocks.append(constructionClock)
    memoryConsumptions.append(memoryConsumption)
    clockAlg.append('Annoy-trees-'+str(numTrees))
    del rez
    del dist
    del result
    gc.collect()

del t
gc.collect()
```