Predviđanje pristajanja malih molekula na proteine korištenjem tehnika umjetne inteligencije na grafičkim procesorima

Tokalić, Adis

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:232330

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetni preddiplomski studij informatike

Adis Tokalić

Predviđanje pristajanja malih molekula na proteine korištenjem tehnika umjetne inteligencije na grafičkim procesorima

Završni rad

Mentor: v. pred. dr. sc. Vedran Miletić

Rijeka, 21. rujna 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 19.02.2020.

Zadatak za završni rad

Pristupnik: Adis Tokalić

Naziv završnog rada: Predviđanje pristajanja malih molekula na proteine korištenjem tehnika umjetne inteligencije na grafičkim procesorima

Naziv završnog rada na eng. jeziku: Protein-small molecule docking prediction using artificial intelligence techniques on GPUs

Sadržaj zadatka:

Porastom procesne moći računala, naročito zahvaljujući računanju na grafičkim procesorima, snažno raste i primjena tehnika strojnog učenja. Jedna od domena primjene strojnog učenja koja se razvija u posljednjih desetak godina je predviđanje molekularnih interakcija. U praktičnoj primjeni vrlo je zanimljivo predviđanje pristajanja malih molekula koje su potencijalni lijekovi na proteine, za što se koriste slobodni softveri otvorenog koda kao što su AutoDock Vina i RxDock i komercijalni softveri kao što je Glide. Cilj rada je dizajnirati analizirati postojeće algoritme strojnog učenja u području predviđanja pristajanja malih molekula na proteine i zatim dizajnirati algoritam koji uči na temelju izlaznih podataka koji daje neki od ranije navedenih softvera pa usporediti predviđanja algoritma za strojno učenje s predviđanjima samog softvera.

Mentor

v. pred dr. sc. Vedran Miletić

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 25.02.2020.

(potpis pristupnika)

Sažetak

Ovaj rad se fokusira na istraživanje metodologije strojnog učenja primijenjene na primjeru pristajanja malih biomolekula na receptore kao što su proteini i nukleinske kiseline. Nakon opisa korištenih testnih podataka, prikazuje se 5 vodećih algoritama, opisuju se njihove funkcionalne karakteristike i uspoređuju njihovi rezultati. Svaki od algoritama je opisan matematički, izveden na dva skupa podataka i zatim uspoređen s ostalim funkcijama. Za dobivanje podataka za učenje i testnih podataka iskorišten je program za predviđanje pristajanja molekula RxDock, a za implementaciju svih algoritama strojnog učenja iskorišten je Python modul sci-kit.

Ključne riječi

strojno učenje, pristajanje molekula, Python, clustering, Bayes, random forest, inteligentni algoritmi, umjetna inteligencija

Sadržaj

1. Uvod....1
2. Osnovne metode strojnog učenja....2
2.1. Testni skupovi....2
2.1.1. D1 testni skup....2
2.1.2. D2 testni skup....3
2.1.3. Skaliranje skupova....3
2.1.4. Raspoređenost skupova: Training i test....4
3. Klasifikacija....5
3.1. Uvod u klasifikacijske algoritme....5
3.2. Vrste klasifikacija....5
3.2.1. Bi-klasifikatori....5
3.2.2. N-klasifikatori....5
3.3. Primjer korištenja klasifikatorskog algoritma....6
3.4. Treniranje klasifikatorskih programa....6
3.4.1. Lazy Learning....6
3.4.2. Eager Learning....7
3.5. Random Forest....7
3.5.1. Načini treniranja....8
3.6. Rezultati RF algoritma....8
3.6.1. Analiza koda RF....12
3.6.2. RandomForestRegressor....13
3.6.3. Rezultati D1 za RF....14
3.6.4. Rezultati za D2 test....15
4. Naivne Bayesove predikcije....17
4.1.1. Naive Bayes....17
4.1.2. Multinominalni Bayes....17
4.1.3. Bernoulijeva-Bayesova metoda....18
4.2. Kodovi za Naive Bayes....18
4.2.1. Bayes kod za D1 skup....18
4.2.2. Bayes kod za D2 skup....19
4.2.3. Karakteristike Bayes algoritama za D1 i D2....19
4.2.4. Rezultati za D1....20
4.2.5. Rezultati za D2....21
5. Regresijski modeli....22
5.1. Matematički koncept regresije....22
5.2. Koeficijenti a0 i a1....24
5.2.1. R squared aproksimacija točnosti....24
5.3. Kodovi za linearnu regresiju....24
5.3.1. Rad korištenih modula....25
5.3.2. Rad glavnog dijela algoritma....26
5.3.3. Rezultati za D1....26
6. Clustering....28
6.1. Vrste clustering algoritma....28
6.1.1. Density-based....28
6.1.2. DBSCAN....28
6.1.3. Hierachical–based....31
6.1.4. Mjerenje clustering algoritma....32
6.2. Kodovi....32
6.2.1. D1 kod za clustering....32
6.2.2. D2 kod za clustering....33

6.3. Objašnjenje koda za clustering....34
6.3.1. D1 Rezultati....35
6.3.2. D2 Rezultat....35
7. Zaključak....36
8. Literatura....37

1. Uvod

Umjetna inteligencija je jedna od najnovijih i najpropulzivnijih grana informatike današnjice. Od regressora do klasifikatora, postoji niz različitih metoda za praktički bilo kakvu primjenu takozvanog strojnog učenja u 21. stoljeću. Ovaj rad je fokusiran na analizu takvih slučajeva uzimajući primjer procjene vezanja atoma [9] i molekula [9]. Naravno, postoji niz različitih podkategorija strojnog učenja, a ovaj rad se fokusira na modele za [5]:

- Klasifikaciju
- Regresiju
- Clustering

Također ovaj rad je namijenjen simuliranju ovih algoritama u radnom okruženju, te zbog toga radimo na realističnom primjeru. Kao glavni primjer za rezultate i funkcionalnost strojnog učenja, algoritmi razvijeni u ovom radu se primjenjuju na biomolekularnu kemiju, specifično na predviđanje spajanja molekule i atoma, kroz razne konformacije. Svi moduli koji su korišteni u ovom radu, uzeti su iz biblioteke **sci-kit** [8] iz nekoliko razloga.

Algoritmi koje koristimo su svi bazirani na strogim uvjetima i dokazima iz matematičke pozadine. To su algoritmi koji su sastavljani godinama i kao rezultat, iza njih stoje godine dokaza, činjenica i izvoda. Kao validni i verificirani algoritmi, određen je najbolji način uporabe, izvođenja i izrade algoritma. Iz tog razloga, ovaj rad je više fokusiran na opisivanje algoritama nego izrade algoritama, te za sve algoritme koristimo **modul Sci-Kit u Pythonu** [8]**, u okolini Jupyter Notebook**.

Sci-kit je modul izrađen primarno za Python koji je specijaliziran u izradi algoritama strojnog učenja, mjerenje metrika takvih algoritama, mjerenje njihove točnosti ,te uspoređivanje njihove brzine i karakteristika. Kao takav, sci-kit [8] je postao standardna platforma za razvijanje algoritama i njihovo testiranje.

U drugom poglavlju opisujemo testne skupove podataka koje ćemo koristiti. Treće, četvrto, peto i šesto poglavlje bave se metodama klasifikacije, naivne Bayesove predikcije, regresije i clusteringa.

2. Testni skupovi podataka

Da bi ovi moduli mogli funkcionirati, potrebno je imati koristan i kvalitetan skup za testiranje. U ovo svrhu razvio sam dvije vrste skupova: **D1** i **D2.** Oni su različiti po konstrukciji, strukturi i funkcionalnosti, a u nastavku ćemo ih daljnje opisati.

2.1. Testni skup D1

	ScoreInter	ScoreIntra	ScoreRestraint	ScoreSystem	ScoreTotal
0	-12.7276	-5.011050	0.0	0	-17.7387
1	-17.7729	-5.338060	0.0	0	-23.1110
2	-24.5639	-0.726793	0.0	0	-25.2907
3	-12.7423	-4.170280	0.0	0	-16.9125
4	-13.8222	-3.689050	0.0	0	-17.5113
...	...	...	...	...	...
95	-27.3283	-2.892940	0.0	0	-30.2213
96	-16.7218	-0.721931	0.0	0	-17.4437
97	-15.2520	-4.271700	0.0	0	-19.5237
98	-15.7291	-3.657500	0.0	0	-19.3866
99	-21.6533	-3.309080	0.0	0	-24.9624

100 rows × 5 columns

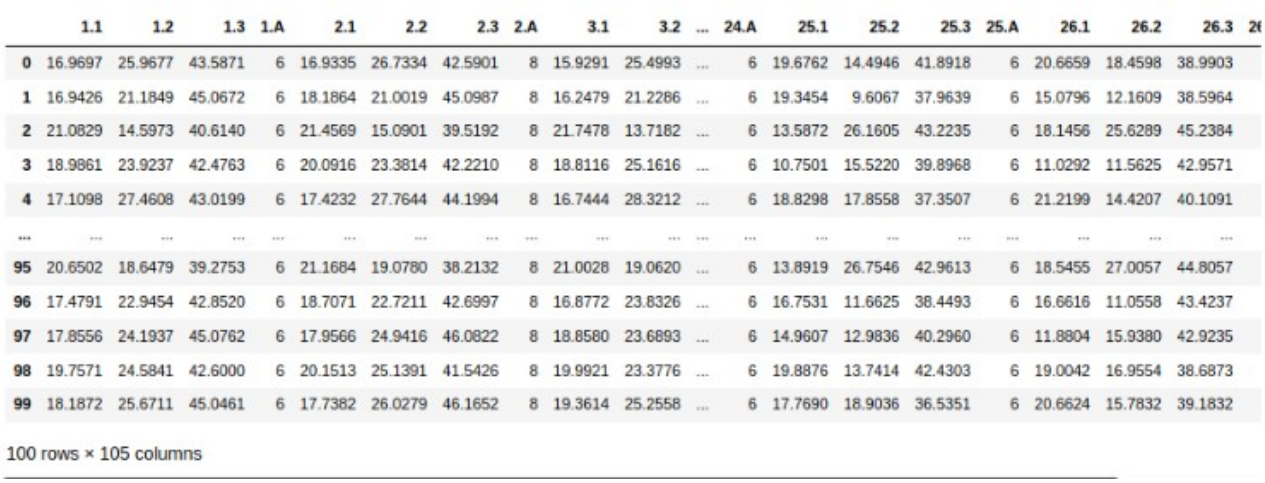
Slika 1: Struktura D1 skupa

D1 se sastoji od 4 glavna stupca: **međumolekularna ocjena** [7]**, unutarmolekularna ocjena** [7], **površna unutarmolekularna ocjena** [7], **ocjena vanjske restrikcije** [7] i glavnog stupca: **ocjena rezultata** [7]. **Ocjena rezultata** se izračunava i aproksimira po ove 4 karakteristike. Karakteristike se baziraju na izračunu programa **RxDock** [7], o kojima možemo dobiti više informacija na službenim stranicama.

Po daljnoj analizi **D1** skupa, možemo vidjeti da su Restraint i System skoro stalno 0, osim u posebnim slučajevima. Daljnja kontekstualna analiza atributa navedenih rezultatova nije potrebna, međutim, bitno nam je da vidimo u kakvoj su relaciji vrijednosti rezultatova međusobno i kako skaliraju jedan s drugim. Ovdje zapažamo da su međusobne vrijednosti dosta varijabilne, bilo to u samo jednom atributu (stupcu) ili gledajući po raličitim karakteristikama. Zbog toga ćemo morati **skalirati**[1] **ove brojčane vrijednosti,** što je objašnjeno kasnije.

Glavni cilj D1 skupa je što točnije procjeniti **ocjena rezultata** [7].

2.2. Testni skup D2



	1.1	1.2	1.3	1.A	2.1	2.2	2.3	2.A	3.1	3.2	...	24.A	25.1	25.2	25.3	25.A	26.1	26.2	26.3	2(
0	16.9697	25.9677	43.5871	6	16.9335	26.7334	42.5901	8	15.9291	25.4993	...	6	19.6762	14.4946	41.8918	6	20.6659	18.4598	38.9903	
1	16.9426	21.1849	45.0672	6	18.1864	21.0019	45.0987	8	16.2479	21.2286	...	6	19.3454	9.6067	37.9639	6	15.0796	12.1609	38.5964	
2	21.0829	14.5973	40.6140	6	21.4569	15.0901	39.5192	8	21.7478	13.7182	...	6	13.5872	26.1605	43.2235	6	18.1456	25.6289	45.2384	
3	18.9861	23.9237	42.4763	6	20.0916	23.3814	42.2210	8	18.8116	25.1616	...	6	10.7501	15.5220	39.8968	6	11.0292	11.5625	42.9571	
4	17.1098	27.4608	43.0199	6	17.4232	27.7644	44.1994	8	16.7444	28.3212	...	6	18.8298	17.8558	37.3507	6	21.2199	14.4207	40.1091	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
95	20.6502	18.6479	39.2753	6	21.1684	19.0780	38.2132	8	21.0028	19.0620	...	6	13.8919	26.7546	42.9613	6	18.5455	27.0057	44.8057	
96	17.4791	22.9454	42.8520	6	18.7071	22.7211	42.6997	8	16.8772	23.8326	...	6	16.7531	11.6625	38.4493	6	16.6616	11.0558	43.4237	
97	17.8556	24.1937	45.0762	6	17.9566	24.9416	46.0822	8	18.8580	23.6893	...	6	14.9607	12.9836	40.2960	6	11.8804	15.9380	42.9235	
98	19.7571	24.5841	42.6000	6	20.1513	25.1391	41.5426	8	19.9921	23.3776	...	6	19.8876	13.7414	42.4303	6	19.0042	16.9554	38.6873	
99	18.1872	25.6711	45.0461	6	17.7382	26.0279	46.1652	8	19.3614	25.2558	...	6	17.7690	18.9036	36.5351	6	20.6624	15.7832	39.1832	

100 rows × 105 columns

Slika 2: Struktura D2 skupa

D2 skup varira od **D1** instantno, brojčano i deskriptivno po podacima. Ovaj skup funkcioira drugačije od prvog, u ovom .csv skupu smo izvukli sve 3D pozicije konformacija [4] za jedan atom po jednom redu. Prva tri stupca su pozicije za prvi smještaj, i onda ide tip atoma. Tip atoma je karakteriziran brojčanom vrijednosću od kojih svaki broj predstavlja jedan elementarni atom. Tablica ima 104 stupca, a zadnji stupac nam je **scoreTotal** [7]. Razlog zašto nismo uključivali gornja 4 faktora u ovu tablicu je zato što ti podaci međusobno bez atribuiranih funkcija nemaju nikakvu karakternu povezanost. Ne možemo doći do navedena 4 rezultata bez da imamo funkciju za računanje. Drugim riječima da njih računamo, uključili bi ih, ali ovdje je cilj samo izračunati **scoreTotal pa pokušavamo prediktirati scoreTotal na navedenim podacima.**

2.3. Skaliranje skupova

Kada pregledavamo podatke u skupovima, vidimo veliku varijaciju brojčane vrijednosti između **D1 i D2 karakteristika,** u svojim omjerima (međusobno u tablicama). Takva disonanca brojeva može dovesti do krivih brojčanih omjera, možemo dobiti visoke brojeve koji reprezentiraju veliku točnost u jednm stupcu u omjeru na drugi skup koji ima male brojeve, a takve male vrijednosti u omjeru na taj stupac također reprezntiraju visoku točnost. Zato skupove međusobno **skaliramo**, kako bi rezultat brojčane vrijednosti bio u omjerima brojeva, pa automatski dobijemo bolju „sliku”

međusobnih odnosa karakteristika.

Tu koristimo **StandardScaler** [8] koji skalira naš skup na omjer minimalnih vrijednosti međusobno između karakteristika kao temelj [11]. Nakon skaliranja, podjelit ćemo testni i trening skup, tako da nam testni bude 40% skupa, a trening 60% skupa. Postotak trening skupa nikad nije apsolutan i može varirati u veličini, međutim za ove testove, 2:3 omjer je davao najbolje rezultate. **StandardScaler** skalira stupce po najmanjoj vrijednosti u stupcu kao početnom vrijednosti [8].

2.4. Raspoređenost skupova: skupovi za treniranje i testni

Svaki skup, **D1** i **D2**, raspoređujemo dalje u dve glavne kategorije: **Test** i **Trening** podskupove [2]. Ovo je ključan korak u većini navedenih algoritama, te je bitno da je kategorizacija nasumična i da ne manipuliramo raspodjelom podataka kako bi replicirali realne uzorke. **Sci-kit** također ima raskošan broj raznih modula za segmentaciju i particioniranje skupova, a u ovom radu, najidealniji je takozvani **train_test_split** [8] modul.

Train_test_split [8] računa sve permutacije segmentiranja danog skupa, a onda iz njih nasumično uzima dve koji imaju međusobno disjunktni presjek(**nijedan uzorak nije u train i testu!)** i sortira ih uz naše određene parametre. On ima nekoliko mogućih parametara za mijenjanje od kojih su najbitniji su:

- test_size- određujemo veličinu test skupa, decimalnom brojkom do 1, gdje bi 0.6 predstavljao 60% cijelog skupa
- train_size – isto kao gore navedeno, bitan je samo jedan od ova dva
- random_state – svaki put kad se splitta skup, nasumično se miješa

Iako pritom možemo manipulirati parametrima, ne možemo manipulirati samom distribucijom uzoraka.

3. Klasifikacija

3.1. Uvod u klasifikacijske algoritme

Klasifikacija [6] je pojam i metoda koja matematički priprema nekakvu funkciju za mapiranje, takozvanu „mapu” koja uzima ulaz X, gleda njegove karakteristike, atribute i sličnosti s drugim ulazima iz testne skupine, te onda mapira neku izlaznu grupu Y. Ovakva metoda je najsličnija matematičkom pojmu funkcije, to jest za svaku nezavisnu ulaznu varijablu X u funkciju f dobijemo dodijeljenu zavisnu varijablu Y [6].

Klasifikacijski algoritmi imaju razne funkcionalne sposobnosti i zbog toga su jedni od najkorištenijih algoritama u pojmu strojnog učenja. Koriste se kada trebamo klasificirati nekakav pojam u jednu od postojećih klasa. Kao rezultat, ne dobivamo novi dodijeljeni uzorak koji je unikatan i može se indetificirati po nekim svojim karakteristikama, nego jedan unos od kolektivnog skupa istih ili sličnih atributa.

3.2. Vrste klasifikacija

Klasifikatori imaju dve glavne podvrste, a to su **bi-klasifikacije** [6] i **n-klasifikacije** [6]**.** Bi-klasifikacije su klasifikatori koji imaju dve glavne grupacije i koriste se kod grube generalizacije klasa. Naprimjer, imamo bi-klasifikator koji određuje pokazuje li se crna boja ili bijela, bi-klasifikator koji označuje prema nekim osobnim podacima „Sviđa mi se” i „Ne sviđa mi se” na određene ulaze, kao filmovi, knjige i ostalo. Kod bi-klasifikatora možemo biti dosta otvoreni prema parametrima pošto imamo omjer grupacija 50%:50%.

3.2.1. Bi-klasifikatori

Bi-klasifikatorske [6] grupacijske skupine su inače velike i dosta obuhvatne, pa imamo više parametara i inače su lakši za implementaciju. Također ovakvi modeli su dosta apstraktni i baziraju se na našem pojmu obuhvatnosti podataka, to jest koliko mi definiramo obuhvatnost neke tematike.

3.2.2. N-klasifikatori

N-klasifikatorski [6] algoritmi su malo kompliciraniji. Kod n-klasifikatorskih algoritama imamo više klasifikacijskih grupacija, pa se parametri i karakteristike jako pažljivo dodjeluju. Za razliku od

bi-klasifikacije treba paziti na zajedničke karakteristike pojedinih skupina, segementacijae karakteristika na detaljnije uzore i dubinom detaljnosti određenih skupina. **N-klasifikatore** [5] je lagano ispodmjernog rasporeda informacijama ali također i prekomjernog rasporeda informacija. Kao rezultat potrebno je puno namještanja i popravljanja danog algoritma, ali baš zbog detaljnosti karakteristika dobivamo dosta točan algoritam. Jedna od danih metoda za olakšavanje građe n-klasifikatorskih algoritama je multi-klasifikacija. Takva metodologija bi imala više klasifikatora podijeljenih na skupine u kojima bi bili tako-zvani pod-klasifikatori. Na taj način bi mogli segementirati grupe s istim filmovima u grupama ali ovisno o atributima, dobili bi detaljnije baziran rezultat.

Ovakvi klasifikatori se najčesće koriste u prepoznavanju govora, rukopisa, biometrijskih indetifikacija, klasifikacija dokumenata, klasifikacija registracijskih tablica na automobilima i sve ostalo što se svodi na klasičnu grupaciju uzoraka.

3.3. Primjer korištenja klasifikatorskog algoritma

Zbog ovog pristupa ovakav algoritam nije idealan za predikciju dockinga molekula na atome, baš zbog toga što ne možemo dobiti izravan i najbolji rezultat. Međutim ovakav pristup nam je pogodan za aproksimacije klasa molekula koje funkcioniraju slično. On više služi za grupiranje molekula, mogao bi se koristiti u biomolekularnoj kemiji za grupiranje kemijskih molekula koje imaju slične kemijske atribute, nekakve supstance koje mogu imati sličan kemijski efekt ili pak čisto klasificiranje letalnosti i opasnosti pojedinih kemijskih tvari. Također, ovakav algoritam bi mogli koristiti za pripremanje idelane uzorne grupe za daljnju eksperimentaciju i provođenje testova biomolekularne tematike.

3.4. Treniranje klasifikatorskih programa

U praksi postoje dve vrste treniranja klasifikacijskih algoritama: lijeno treniranje i brzo treniranje [5].

3.4.1. Lazy Learning

Lijeno treniranje ili **„Lazy Learning”** [5] je treniranje koje klasificira testni uzorak nakon spremanja uzorka za treniranje. Ovi algoritmi troše manje vremena na treniranje uzorka ali više na predikciju i klasifikaciju novi nepoznatih uzorka. Ovakvi algoritmi su napravljeni da pouzdano rade

u nepoznatoj okolini s nepoznatim podacima. Ako nam algoritam radi s poznatim podacima, koristimo **„Eager Learning”**. Primjer ovakvog algoritma je **K-nearest neighbour**.

3.4.2. Eager Learning

„Eager Learning” [5] algoritmi su algoritmi koji klasificiraju bez testnih podataka i troše više komputacijskog vremena na treniranje a manje vremena na samo prediktiranje novih uzoraka. Ovakvi algoritmi se koriste za neke uzorke za koje imamo određene testne i trening podatke i bitnije nam je da taj sustav radi bolje s poznatim podacima. Ovakav sustav se koristi u poznatoj okolini i njegov glavni cilj nije da uči nego da radi s već pripremljenim podacima. Primjer ovakvog algoritma bi bio **ANN**, **Naivni Bayes** i **Decision Tress**(**RF**, **Boosted**) [11].

3.5. Random Forest

RF [5] ili random forests je jedan od klasifikatorskih metoda koje koristimo da iztreniramo klasifikatorske algoritme. Algoritam se bazira na pristupu klasifikatorskih stabala, to jest **classification trees-a.** To su klasifikacijski grafovi koji se koriste za grupaciju. Svako **stablo** ima svoj **korijen**, i kada podatak ulazi u klasifikaciju, po čvorovima uspoređujemo atribute čvora i ulaznog podataka, te ulazni podatak putuje po čvorovima koji su najsličniji njegovim atributima. Nakon nekog vremena ulazni podatak dolazi do kraja stabla i tamo se klasificira s najsličnijim čvorom stabla. **Za klasifikacijsko stablo su nam potrebni sljedeći podaci:**

- **Dimenzija stabla** [4] – ovo je faktor koji odlučuje maksimalnu dubinu stabla – y varijablu stabla
- **Pravilo** [4] – Ovo je atribut po kojem mi gledamo kako će se micati čvor u stablu. Najčesće se ovaj atribut skalira u numeričku vrijednost od 0 do 1, te ako je pravilo iznad neke određene vrijednosti, npr. 0.50, podatak odlazi u desni čvor, a u suprotnom, podatak odlazi u lijevi čvor. Ako imamo više atributa, moguće je skalirati sve atribute u jedno pravilo, neki standardnim scalerom i time sažeti ih u jedan broj**(Ovo radimo kod klasifikacije s više atributa)**
- **Lijevi i desni čvor** [4]– U svakom koraku, podatak mora znati raspoznati lijevi i desni čvor ispred njega tako da može funkcionalno hodati kroz čvorove stabla i samo stablo
- **Klasifikacije** – Podatak mora ima kategorije lijevog i desnog čvora, tako da se zna koje su klasifikacije pod njima

Broj stabla se podešava ručno kako bi se precizno definirala najveća točnost. Podaci koji idu u svako stablo su nasumično odabrani, ali broj podataka je isti. **Ovo se radi da ne bi imali „namještene" rezultate** [8]. Cilj ovog algoritma je da radi u poznatom okruženju s poznatim podacima, i da dobijemo stabla koja kategoriziraju međusobno drugačije ali ih je moguće podkategorizirati(stabla) u neke grupacije. **U drugim riječima bitno je da imamo sličnosti po stablima** [8].

3.5.1. Načini treniranja

Random Forest algoritam [8] uzima načelo normalnih decision treesa(klasifikacija po stablima, ne mora biti nasumična) i **izrađuje n nasumičnih stabala**. Svako stablo dobiva nasumični skup istobrojnih podataka. Svako stablo se trenira pojedinačno, te zbog toga je ovaj algoritam najbolje trenirat preko paralelizma **GPU** komputacije. Postoje dva načina **paralelne obrade** i treniranja podataka:

- **Stabla se može trenirati individualno u isto vrijeme** [8], komunikacija između stabala ne postoji pa se svako stablo trenira po svojoj određenoj brzini
- Ulazi se izračunavaju paralelno po dubinama. Ovdje postoji komunikacija između stabala pošto se svaka dubina trenira istovremeno **(malo kompleksniji način, mora se implementirat komunikacija među stablima)**

Korak po korak algoritma na paralelnoj obradi [4]:

- **CPU:1** dodjeluje nasumične istobrojne podatke stablima i kopira na GPU
- **CPU:2** Za svako stablo, razvij čvorove da se mogu dodjeljivati i pošalji zahtjeve na GPU
- **GPU:3** Treniraj početne čvorove dok entropija nije minimalna, ili je ispod zadovoljavajućeg kriterija. Ovakav način se može iterativno trenirati dok ne dobijemo zadovoljavajuću preciznost
- **CPU:4** Kopiraj requestove s GPU-a. Ako čvor postoji, formuliraj zahtjev na GPU, za smještanje daljneg ulaznog podataka, dok se ne izgradi stabla

3.6. Rezultati RF algoritma

RF algoritam je izgrađen na sljedeći način. Korištena je sci-kit biblioteka za Random Forest

Reggressor, numpy i pandas biblioteke za rad sa .csv datotekama, moduli iz sklearna(pod modul Sci-Kita), preciznije **train_test_splitter** [8], **Scaler** [8], i **metrics** [8] Također je korišten **matplotlib** za plottanje daljnjih rezultata.

```
#Importanje svih potrebnih biblioteka
import pandas as pd
import numpy as np
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
import matplotlib.pyplot as pl

#D1 tablica
podaci= pd.read_csv('/home/adis/Desktop/RF/D1.csv')

#prvih 100 podataka
podaci.head(100)

#X vrijednosti i Y vrijednosti
X = podaci.iloc[:, 0:4].values
y = podaci.iloc[:, 4:].values
print(X)
print(y)

#X,Y skupovi za treniranje i testiranje
X_tren, X_testing, y_tren, y_testing = train_test_split(X, y, test_size=0.4 ,random_state
=0)

#scaler za X trening i test skupove da bi imali sličnije brojeve i bolje rezultate

sc = StandardScaler()
X_tren = sc.fit_transform(X_tren)
X_testing = sc.transform(X_testing)


#Regressor i određivanje prediktora
reg = RandomForestRegressor(n_estimators=500, random_state = 0)
reg.fit(X_tren, y_tren)
y_predik = reg.predict(X_testing)
y_treningpredik =reg.predict(X_tren)

#Accuracy za 2hr1

print('Prosjecna aboslutna pogreska:', metrics.mean_absolute_error(y_testing, y_predik))
print('MSE:', metrics.mean_squared_error(y_testing, y_predik))
```

```
print('MSE Korijen:', np.sqrt(metrics.mean_squared_error(y_testing, y_predik)))


pl.plot(X_tren, y_tren, 'o')

pl.xlabel("X skup Trening")
pl.ylabel("Y skup Trening")
axes = pl.gca()
axes.skup_xlim([-5,5])
axes.skup_ylim([-40,20])
pl.show()

pl.plot(X_testing,y_testing, 'o')
pl.xlabel("X Test skup")
pl.ylabel("Y Test skup")

axes = pl.gca()
axes.skup_xlim([-5,5])
axes.skup_ylim([-40,20])

pl.plot(y_tren, y_treningpredik,'ro', y_testing,y_predik, 'o' )
pl.xlabel("Y Stvarna")
pl.ylabel("Y Prediktirana")
axes = pl.gca()
axes.skup_xlim([-25,-40])
axes.skup_ylim([-25,-40])
pl.legend(['Train podaci', 'Test podaci'])


pl.plot(y_treningpredik, y_tren ,'o' )
pl.xlabel("Stvarni Test skup 2hr1 ")
pl.ylabel("Prekditirani Test skup 2hr1")
axes = pl.gca()
axes.skup_xlim([-25,-40])
axes.skup_ylim([-25,-40])
```

Ovaj specifični algoritam je napravljen za **D1 testne podatke**, dok je D2 modificiran za sebe posebno:

```
#Importanje svih potrebnih biblioteka
import pandas as pd
import numpy as np
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn import metrics
import matplotlib.pyplot as pl

#d2 tablica
podaci= pd.read_csv('/home/adis/Desktop/RF/d2.csv')

#prvih 100 podataka
podaci.head(100)

#X vrijednosti i Y vrijednosti
X = podaci.iloc[:, 0:104].values
y = podaci.iloc[:, 104:].values
print(X)
print(y)

#X,Y skupovi za treniranje i testiranje
X_tren, X_testing, y_tren, y_testing = train_test_split(X, y, test_size=0.4
,random_state=0)

#scaler za X trening i test skupove da bi imali sličnije brojeve i bolje rez
ultate

sc = StandardScaler()
X_tren = sc.fit_transform(X_tren)
X_testing = sc.transform(X_testing)


#Regressor i određivanje prediktora
reg = RandomForestRegressor(n_estimators=500, random_state = 0)
reg.fit(X_tren, y_tren)
y_predik = reg.predict(X_testing)
y_treningpredik =reg.predict(X_tren)

#Accuracy za 2hr1

print('Prosjecna aboslutna pogreska:', metrics.mean_absolute_error(y_testing
, y_predik))
print('MSE:', metrics.mean_squared_error(y_testing, y_predik))
print('MSE Korijen:', np.sqrt(metrics.mean_squared_error(y_testing, y_predik
)))


pl.plot(X_tren, y_tren, 'o')

pl.xlabel("X skup Trening")
pl.ylabel("Y skup Trening")
axes = pl.gca()
axes.skup_xlim([-5,5])
axes.skup_ylim([-40,20])
```

```
pl.show()

pl.plot(X_testing,y_testing, 'o')
pl.xlabel("X Test skup")
pl.ylabel("Y Test skup")

axes = pl.gca()
axes.skup_xlim([-5,5])
axes.skup_ylim([-40,20])

pl.plot(y_tren, y_treningpredik,'ro', y_testing,y_predik, 'o' )
pl.xlabel("Y Stvarna")
pl.ylabel("Y Prediktirana")
axes = pl.gca()
axes.skup_xlim([-25,-40])
axes.skup_ylim([-25,-40])
pl.legend(['Train podaci', 'Test podaci'])


pl.plot(y_treningpredik, y_tren ,'o' )
pl.xlabel("Stvarni Test skup 2hr1 ")
pl.ylabel("Prekditirani Test skup 2hr1")
axes = pl.gca()
axes.skup_xlim([-25,-40])
axes.skup_ylim([-25,-40]
```

3.6.1. Analiza koda RF

Pri pokretanju koda, uvoze se dane biblioteke. Neke od biblioteka su već objašnjenene na početku, ali eksluzivni moduli za RF algoritam su:

- Train_test_split [8]
- StandardScaler [8]
- RandomForestRegressor [8]

Train_test_split [8] je modul iz sklearn.model_selection biblioteke koji je odgovoran za kreiranje testnih i trening skupova. Ovo je modul koji koristimo u većinu programskih kodova strojnog učenja, zbog efikasnosti grupacije ulaznih podataka. Potpuno je promjeniv i skalabilan za svaki proces.

StandardScaler [8] je algoritam koji koristimo za skaliranje podataka. Naši ulazni podaci mogu varirati brojčano u vrijednostima, te se može dogoditi da imamo velike prosječne razlike, pošto je velika međusobna razlika i odstupanje između našeg ulaznog skupa. Da bi dobili što točniju

predikciju i bolji opis međusobnih veza između naših ulaznih vrijednosti, skaliramo ih na neke manje brojeve, najčesće od 0 do 1 najmanji podatak, a od 1 do 3, ostale podatke.

Na primjer: 1.50 bi bio podatak koji je 50% veći od ili 150% prosječnog ulaznog podatka.

3.6.2. RandomForestRegressor

Random Forest Reggresor je glavna klasa koju koristimo za naš algoritam. Njegova funkcionalnost je objašnjena u članku iznad, ovdje ćemo se fokusirati samo na atribute naše klase. RFR ima sljedeće parametre („Preuzeto iz službene dokumentacije”):

N_estimators [8] -> broj treeova, u forestu

Criterion [8] ->funkcija koja mjeri točnost algoritma, uobičajena funkcija je **MSE[1]**

Max_depth [8] -> maksimalna dubina stabla

Min_samples_leaf [8] –> Najmanji broj čvorova koji su listovi

Max_features [8] -> Maksimalan broj atributa nekog čvora koji se gledaju pri korištenju algoritma

Algoritam je podijeljen u nekoliko sekvenca. Za D1 i D2, prvo učitivamo .csv tablicu s podacima. Nakon učitanih podataka, imamo opciju da ispišemo prvih 100 podataka, samo da provjerimo je li input bio dobar, ali također možemo pogledati karakteristike naših podataka. Kod naših podataka, vidimo ogromnu varijaciju u prostoru. Ovdje koristimo već spomenuti **StandardScaler** [8] kako bi skalirali tablicu i njene karakteristike

Kada skaliramo i raspodjelimo skupove, onda koristimo regresijsku random forest klasu da iztreniramo naš RF algoritam. Koristit ćemo 500 estimatora, to jest 500 stabala, u koje onda sortiramo prvo testne podatke da naš algoritam nauči sortirati podatke, a onda stavljamo trening podatke i gledamo rezultate.

3.6.3. Rezultati D1 za RF

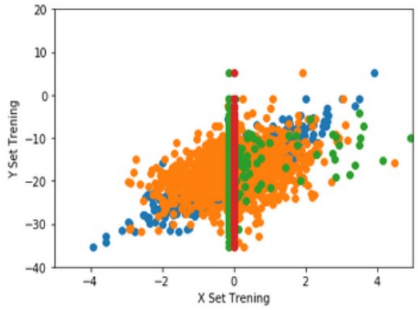
```
In [8]: #Accuracy za 2hr1

        print('Prosjecna aboslutna pogreska:', metrics.mean_absolute_error(y_testing, y_predik))
        print('MSE:', metrics.mean_squared_error(y_testing, y_predik))
        print('MSE Korijen:', np.sqrt(metrics.mean_squared_error(y_testing, y_predik)))

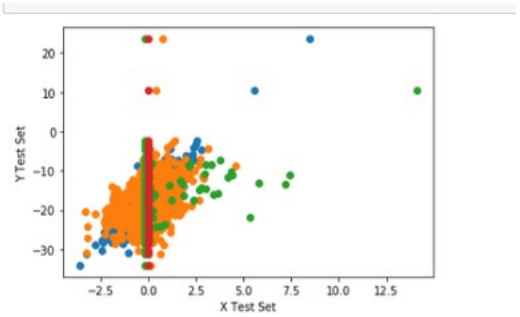
Prosjecna aboslutna pogreska: 0.21699424240636464
MSE: 0.9965196771766087
MSE Korijen: 0.9982583218669447
```

Slika 3: Testiranje i rezultati za D1 RF

Pri testiranju D1 skupa, dobili smo jako točne rezultate. Prosječna apsolutna pogreška je bila oko 0.2, kao što se vidi u skupu te pogreške nisu bile česte, ali one koje se jesu dogodile, su bile dosta velike, povečavajući prosječnu grešku. **MSE** [1] **je ispod 0.99**, što je glavni indikator da ovaj model je dobro fittan, a to se i može zaključiti jer se gledaju samo 4 indikatora tj 4 karakteristike i 1 target atribut koji se prediktira.

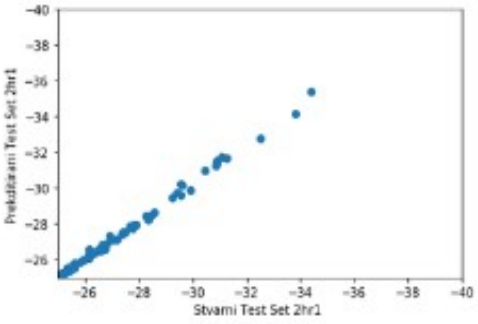


Slika 5: Mapiranje Trening skupa D1 RF

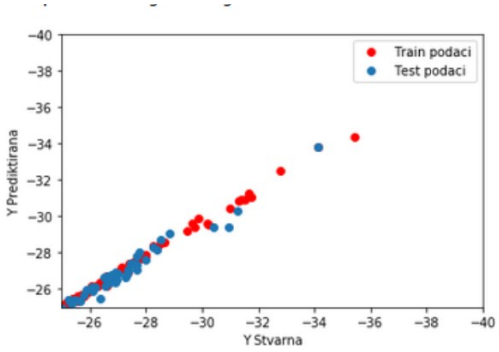


Slika 4: Mapiranje Test skupa D1 RF

Kod mapiranja testnog i trening skupa možemo zaključiti sljedeće: **Algoritam** [5] **ima očitu pristranost prema negativnim vrijednostima u test skupu** [5]. Ova anomalija je vjerovatno rezultirana činjenicom da su nam sve vrijednosti rezultatova u negativnom prostoru. Međutim pri treniranju, algoritam ne zna tu činjenicu i distribuira trening skup prema svojojoj bazi estimatora tj stabala. Kako algoritam prolazi kroz nekoliko epoha, počinje razumijevati da trenira samo an negativnim vrijednostima i nakon 5 epoha dobivamo balans pozitivnih i negativnih vrijednosti. Međutim, interesantna je činjenica što na Test skupu ovaj algoritam ispravno smješta sve negativne vrijednosti u negativni prostor, ali zadržava još uvijek balans. Omjer negativnih i pozitivnih rezultatva je 60%-40%.



Slika 6: Mapiranje pravih i aporksimiranih vrijednosti za Test skup D1 RF



Slika 7: Mapiranje Pravih i aproksimiranih vrijednosti za trening skup D1 RF

Na slikama mapiranja prediktiranih vrijednosti sa stvarnima, i test i train podaci su visoke točnosti. Ovdje vidimo kako postoji linearna ovisnost u predikciji jer je većina podatak točno aproksimirana.

3.6.4. Rezultati za D2 test

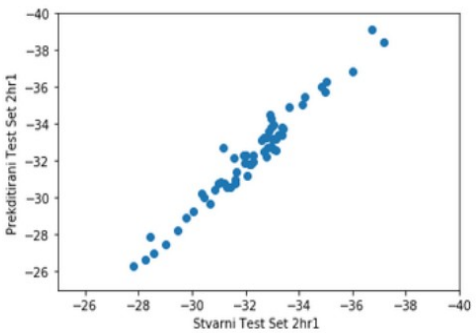
```
#Accuracy za 2hr1

print('Prosjecna aboslutna pogreska:', metrics.mean_absolute_error(y_testing, y_predik))
print('MSE:', metrics.mean_squared_error(y_testing, y_predik))
print('MSE Korijen:', np.sqrt(metrics.mean_squared_error(y_testing, y_predik)))
```

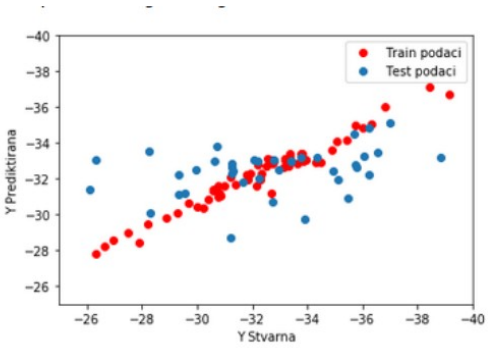
```
Prosjecna aboslutna pogreska: 2.433564255000004
MSE: 9.36641920075542
MSE Korijen: 3.06046061904992
```

Slika 8: Rezultati za D2 skup RF

Kod podataka u D2 skupu, dobivamo znatno lošije rezultate. **MSE** [1] **je ekstremno visok, što sugestira da smo prekomjerno rasporedili uzorke u model** . Zbog načina koji je ova csv tablica napravljena, velika je vjerojatnost da je model **prekomjerno rasporedeđen** i da smo pretrenirali naš algoritam.



Slika 10: Mapiranje pravih i aproksimiranih vrijednosti za D2 RF Test skup



Slika 9: Mapiranje pravih i aproksimiranih vrijednosti D2 Trening skupa za RF

Kada gledamo aproksimacije u omjeru na stvarne testne i trening vrijednosti, vidimo da je test skup relativno dobro iztreniran u nekim epohama. Rezultati ogromno variraju i to se može vidjeti na sljedećim slikama.

4. Naivne Bayesove predikcije

NB algoritmi se baziraju na Bayesovom teoremu, s predpostavkom da su svi prediktori karakteristično disjunktni jedan s drugim. To znači da u našoj klasifikaciji, moramo segregirati sve zavisne prediktore i gledati ih na nezavisan način. Jedan od načina je da skaliramo 3D koordinate molekula, to jest X,Y i Z u jednu varijablu, tako postupimo s ostalim prediktorima dok su svi skalari koje imamo od 0 do 1 i nisu međusobno ovisni.

Ovakva predikcija se bazira na činjenici da pronalazimo postotak dodjeljivanja neke grupacije po posmatranju nekih faktora. Ovo se radi kroz Bayesov teorem, to jest formulu

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Drugim riječima gledamo postotak ili šansu da se Klasifikacija A dobije iz cijele skupine grupacija B. Za ovakvu računicu, potrebno nam je nekoliko stvari:

- Postotak da će bilo koja grupacija iz B izvući tako da je moguće da je jedna od tih grupacija A
- Postotak da dobijemo jednu od tih grupacija iz cijele klasifikacijske grupe koja zadovoljava uvjete
- Postotak da će se izvući A iz cijele grupacijske skupine(B ne mora uvijek biti cijela skupina, nego samo oni koji zadovoljavaju uvjete)

4.1.1. Naive Bayes

Ovakav model se bazira na mješavini Gaussovih distribucija. Svaka grupacija je jednostavna Gaussova distribucija. **Gaussova krivulja** [1] **ili funkcija** izražava vjerovatnost da će se bilo koje posmatranje uzorka naći između dvije stvarne granice. Krivulja s obije strane teži k nuli, pa ovom distribucijom mapiramo sva posmatranja na neku određenu krivulju.

Na ovakav način sve klasifikacije su nam zapravo samo krivulje koje su mapirane na vrijednosti uzorka u toj grupaciji.

4.1.2. Multinominalni Bayes

Distribucija je slična Gaussovoj međutim, sve klasifikacijske grupacije su mapirane po

multinominalnoj distribuciji. Multinominalna distribucija [1] je generalizirana binomna distribucija, gdje imamo skup uspjeha i neuspjeha n eksperimenata i svaki može imat svoj booleanski ishod(T/F,0/1 itd..). Generaliziramo li binomnu distribuciju, tako da imamo n binomnih distribucija a svaki i=1,.....,n predstavlja distribuciju jedne klasifikatorske grupe. Ovakav submodel Bayesa je najbolji za diskrentno brojanje svake klasifikacije.

4.1.3. Bernoulijeva-Bayesova metoda

Zadnja metoda građena Bayesovim modelom je Bernoulijeva-Bayesova metoda. Što čini ovu metodu dosta interesantnom je da su svi feature-i uzorka su binarne varijable(0 ili 1): Ovakav klasifikacijski model je najbolji za **document classification [1]** ili bilo kakvu drugu klasifikaciju gdje imamo čiste individualne i unikatne atribute u uzorku grupacije pa ga generalno ne koristimo u svrhe za predikciju i mapiranje dockinga molekula i atoma. Primjer bi također bi klasifikacija nekih kratkih tekstova, pošto radi jako dobro na minimalnim ne proširenim uvjetima.

4.2. Kodovi za Naive Bayes

4.2.1. Bayes kod za D1 skup

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as pl

podaci = pd.read_csv('/home/adis/Desktop/RF/D1.csv')

X = podaci.iloc[:, 0:4].values
y = podaci.iloc[:, 4].values

enkoder = LabelEncoder()

y = enkoder.fit_transform(y)

X_tren, X_testing, y_tren, y_testing = train_test_split(X,y, test_size=0.20,
random_state=1)


sc = StandardScaler()
X_tren = sc.fit_transform(X_tren)
```

```
X_testing = sc.transform(X_testing)


clf = GaussianNB()
clf.fit(X_tren, y_tren)
GaussianNB(priors=None)
y_predik = clf.predict(X_testing)


pl.plot(X_testing,y_testing, 'o')
pl.xlabel("X test")
pl.ylabel("Y test")

axes = pl.gca()
```

4.2.2. Bayes kod za D2 skup

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as pl

podaci = pd.read_csv('/home/adis/Desktop/RF/d2.csv')

X = podaci.iloc[:, 0:4].values
y = podaci.iloc[:, 4].values

enkoder = LabelEncoder()

y = enkoder.fit_transform(y)

X_tren, X_testing, y_tren, y_testing = train_test_split(X,y, test_size=0.20,
random_state=1)


sc = StandardScaler()
```

4.2.3. Karakteristike Bayes algoritama za D1 i D2

Slično kao i u RF algoritmu, postoji nekoliko glavnih biblioteka koje smo već objasnili, ali i nove biblioteke eksluzivne za ovaj algoritam. Najvažnije su:

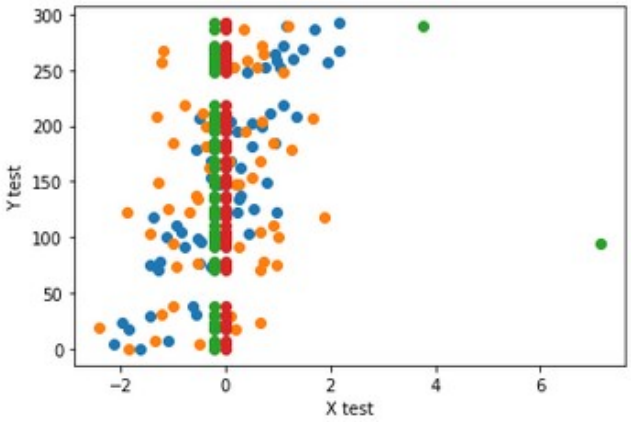
```
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
```

- **sklearn.naive_bayes** nam daje GaussianNB koji sadrži Bayes-Gaussian algoritam. On funkcionira kao gore navedeno, a njegovi glavni parametri su:
- **priors** [8] - n-nizovi koji opisuju vjerojatnosti svakog uzorka u nekoj distribuciji
- **var_smoothing** [8] - uklanja takozvani „noise" iz skupova podataka.

Nakon određivanja X i Y atributa, koristimo enkoder koji labelira naše y funkcije tako da svaka bude unikatna, a labelira brojevima od 0 do n-1. Na ovaj način nemamo dve slične funkcije po vrijednostima, cilj ovog algoritma nije regresija nego čisto grupiranje. Onda određujemo testnu i trening skupinu, koristimo scalere da skaliramo skupove i **kreiramo GaussianNB** [8] **algoritam.** On raspodjeljuje trening skup u gaussove distribucije i radi nove vjerojatnosti za svaki skup(priors=none je tu jer nemamo već psotojeće vjerojatnosti nego ih on kalkulira).

```
GaussianNB(priors=None)
```

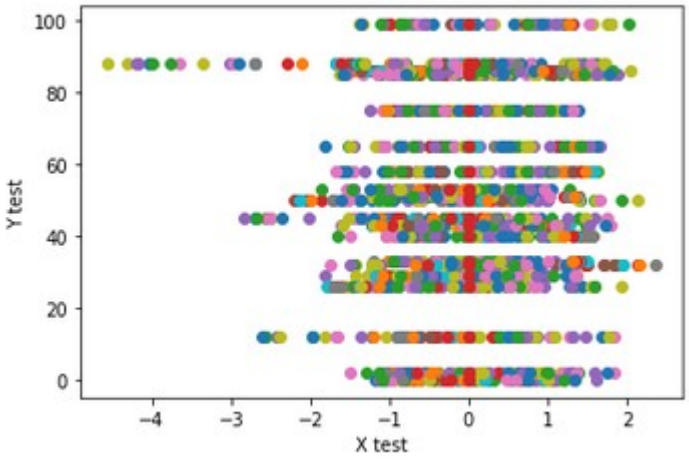
4.2.4. Rezultati za D1



Slika 11: Rezultati za Bayesa za D1

Rezultati dobiveni ovom metodom su posebno interesantni. Za **D1** je vrlo očito da zbog 3. i 4. stupca koji su skoro na svim ulazima 0, imamo stupac za te dve karakteristike u X=0. Ostali uzorci su, međutim, dosta kaotično grupirani. Ne vidimo nikakve grupacije clustera ili fokusirane grupe jedne karakteristike pa možemo zaključiti da rezultatovi nemaju međusobnu korelaciju [2].

4.2.5. Rezultati za D2



Slika 12: Rezultati D2 za Bayesa

Na prvi pogled kod ovih rezultata vidimo da su očito grupirani. Iako imamo 104 međusobne karakteristike, postoje očitii clusteri u redovima, specifičnije postoji točno onoliko clustera koliko ima sličnih konformacija. Ovakav pristup rezultira u jako bitnim informacijama jer možemo provesti daljnje grupiranje po clusterima [2].

5. Regresijski modeli

5.1. Matematički koncept regresije

Regresijski modeli [1] su jedni od najkorištenijih modela za treniranje i jedan od najboljih modela za predikcije dockinga molekula i atoma. Cijeli koncept se bazira na matematičkom konceptu regresije a analizira ovisnu varijablu kao rezultat funkcije i njen odnos s neovisnom varijablom. Matematička reprezentacija ovisnosti ovisne varijable o neovisnoj se može izraziti kao:

$$Y = aX + b[2]$$

Y je ovisna varijabla o **X** koji je neovisan i može se odrediti svojevoljno, **a** je koeficijent regresije koji određuje koji i kakav efekt X ima na Y, a b je konstanta. Naprimjer ako bi X bio 0, onda bi za Y vrijedilo

$$Y = b^2[2]$$

Regresija je metoda koja modelira vrijednost uzorka u skupove po neovisnim varijablama i cilj ovakve metode u strojnom učenju nam je da odredi nekakvu vrijednost koja bi bila najbliža i najrealističnija rezultatu koju bi dobili da uvrstimo takav uzorak u navedeni događaj. Linearna regresija je vrsta regresije gdje je vrsta ovisnosti neovisne i ovisne varijable linearna. Pokušavamo pronaći idealni pravac koji prolazi između svih danih uzoraka . Y izračunavamo po linearnom koeficijentu koji glasi:

$$\boldsymbol{Y = a_0 + a_1 x[2]}$$

Glavni cilj rješavanja algoritma linearne regresije je pronalaženje koeficijenata $\mathbf{a_0}$ [1] i $\mathbf{a_1 x}$.[1] Kako nam treba najpovoljnija i najbolja vrijednost navedenih elemenata, pokušavamo minimizirati navedene koeficijente. Jedan od načina kako bi dobili najbolje vrijednosti je da uzimamo udaljenosti od pravih točki, to jest njihovih pravih vrijednosti i točki koje aproksimiramo. Što je manja udaljenost, to znači da je aproksimacije točnija. Formula az udaljenost glasi:

$$d_i = \sqrt{(x_i - x_i)^2 + (y_{stvarna} - y_{aproksimirano})^2}[2]$$

$y_{aproksimirano}$možemo odrediti tako da ubacimo u funkciju:

$$y_{aproksimirana} = f(x_i)[2]$$

Kroz ovu logiku najbolja funkcija je ona koja nam ima najmanju sumu svih udaljenosti to jest:

$$\sum_{i=0}^{n} d_i[2]$$

Također se koristi suma najmanjih kvadrata jer ju je uodređenim slučajevima znatno lakše odrediti. Kroz ovakav pristup problematiku minimalne sume možemo odrediti kao

$$y-(a_0+a_1x)^2[1]$$

Y[1] – stvarna vrijednost za **X** varijablu

a_0+a_1x [1] – Aproksimirana vrijednost za **X** varijablu

Naravno, ovo je čisti matematički pristup, u praksi kod strojnog učenja koristimo:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \quad [5]\ [8]$$

Te je kao **MSE[1](Mean Squared Error**). Prvo uzimamo grešku predikcije za svaki pojedinačni uzorak, kvadriramo i napramo sumu svi grešaka predikcije koji dijelimo s brojem uzorka. Rezultat je prosječna kvadratna greška to jest prosječno odstupanje uzoraka. Pomoću MSE-a možemo mijenjati koeficijente $a_0 i\, a_1$ te uzeti minimalnu vrijednosti MSE-a kako bi dobili najtočniju aproksimaciju algoritma.

Najbitnija karakteristika kod regresijskog modela je postupno otpadanje. Koncept postupnog otpadanja je da uzmemo $a_0 i\, a_1$ neke vrijednosti za neku točku I iterativno ih mijenjamo da smanjimo trošak i povećamo točnost algoritma. Broj iteracija u postupnom otpadanju je brzina učenja algoritma, te ona određuje koliko algoritam brzo konvergira do minimalne vrijednosti. U drugim riječima da bi ažurirali $a_0 i\, a_1$ svaki iteraciju posebno, uzimamo gradijent za sve točke funkcije u toj iteraciji. Da bi našli gradijent moramo naći djelomične derivacije svake točke za $a_0 i\, a_1$.

Iz **MSE** [1] izračuna možemo doći do derivacije po $a_0 i$ *derivacije po* a_1 . Kada deriviramo djelomićno po svakom koeficijentu dobijemo sljedeće jednažbe:

$$\frac{\partial V}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0+a_1*x_i-y_i)^2 \qquad \frac{\partial V}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0+a_1*x_i-y_i)^2*x_i$$

5.2. Koeficijenti a0 i a1

Iz ovoga izračuna dobivamo konačne formule za koeficijente:

$$a_0 = a_0 - \frac{\propto * 2}{n} \sum_{i=1}^{n} (a_0 + a_1 * x_i - y_i)^2 \qquad a_1 = a_1 - \frac{\propto * 2}{n} \sum_{i=1}^{n} (a_0 + a_1 * x_i - y_i)^2 * x_i$$

$\propto$ [2] je brzina učenja našeg algoritma. On mora biti specifiran je ovisno o njegovom iznosu možemo imati bržu ili sporiju ratu učenja. Veći $\propto$ može dovesti do bržeg ali ne precizinjeg algoritma dok manji $\propto$ [2] može dovesti do sporijeg ali puno efikasnijeg algoritma. Naravno, utjecaj $\propto$ varira o kompleksnosti skupa podataka, njihovim parametrima i entropiji sustava.

5.2.1. R squared aproksimacija točnosti

Za mjerenje sustava ćemo koristiti R-squared metodu mjerenja – $\mathbf{R^2}$ [2]. $\mathbf{R^2}$mjerenja su prilično direktna, a to nije ništa drugo nego koeficijent obzervirane varijacije kroz broj totalnih varijacija. $\mathbf{R^2}$ varira od 0 do 100, gdje je 0 opisuje model nikakve varijacije povratnih uzoraka, to jest algoritam je ne učinkovit i ne primjeniv, kategorizira i predikitra sve uzorke na jednu točku. 100% $\mathbf{R^2}$ opisuje algoritam koji ima 100% varijaciju na povratnom skupu što znači da jako dobro estimira i svaka testna točka ima svoju estimaciju. Generalno se traži model s $\mathbf{R^2}$ **koji teži prema vrijednosti od 100, ali postoji par anomalija** [2].

5.3. Kodovi za linearnu regresiju

Za linearnu regresiju koristili smo samo D1 skup zbog nepredvljivih I ne optimiziranih rezultata [1] od skupa D2. Zbog previse karakteristika I atributa bez ikakvih funkcijskih relacija, linearnu regresiju nije bilo moguće napravit na D2 skupu. Kod za D1 izgleda ovako:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import dataskups, linear_model, metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as pl

podaci = pd.read_csv('/home/adis/Desktop/RF/D1.csv')

podaci.head(100)
```

```
X = podaci.iloc[:, 0:4].values
y = podaci.iloc[:, 4:].values
print(X)
print(y)



X_tren, X_testing, y_tren, y_testing = train_test_split(X, y, test_size=0.7
,random_state=0)

reg = linear_model.LinearRegression()
reg.fit(X_tren, y_tren)
print('CKoeficijenti \n', reg.coef_)
print('Varijanca {}'.format(reg.rezultat(X_test, y_test)))
plt.style.use('fivethirtyeight')
plt.scatter(reg.predict(X_tren), reg.predict(X_tren) - y_tren, color = "red"
, s = 10, label = 'Train podaci')
plt.scatter(reg.predict(X_testing), reg.predict(X_testing) - y_testing, colo
r = "green", s = 10, label = 'Test podaci')
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)
plt.legend(loc = 'upper right')
plt.show()

pl.plot(X_tren, y_tren, 'o')

pl.xlabel("X trening")
pl.ylabel("Y trening")
axes = pl.gca()

pl.show()
```

5.3.1. Rad korištenih modula

Do sada smo naveli i opisali sve bilbioteke koje se ponavljaju u ovim testovima, a posebne za regresiju praktički nemamo. Imamo neke nove module a to su:

- **dataskups** [5], **linear_model** [5] i **metrics** [5] **–** Datatskups modul koristimo za

formatiranje i estetičko formiranje podataka, dok je linear model zapravo modul koji je odgovoran za linearno regresijski model koji imamo u našem algoritmu. Metrics je modul koji pomaže s radom oko metrika i mjernih jedinica (njihovih konverzija, etc.)

5.3.2. Rad glavnog dijela algoritma

Prošli smo već sve poznate module i znamo kako formiramo naše skupove,a regresiju smo i na neki način prošli kod RF-eva. Naime, ovo je algoritam koji je jednostavan za implementirat i cijelo regresijsko namještanje nam se održava u sljedećoj liniji:

```
reg = linear_model.LinearRegression()
reg.fit(X_tren, y_tren)
```

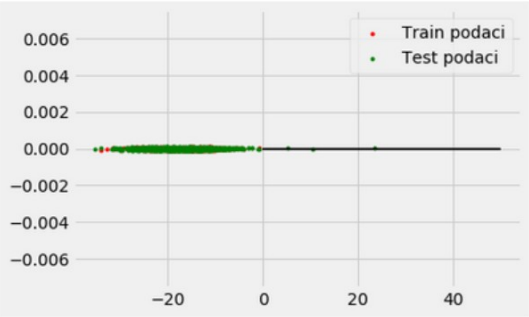
Nakon što instanciramo linearni regressor, moramo ga samo fittat s podacima.

5.3.3. Rezultati za D1

```
CKoeficijenti
 [[1.00000064 1.00000106 0.99998028 0.        ]]
Varijanca 0.9999999999390922
```

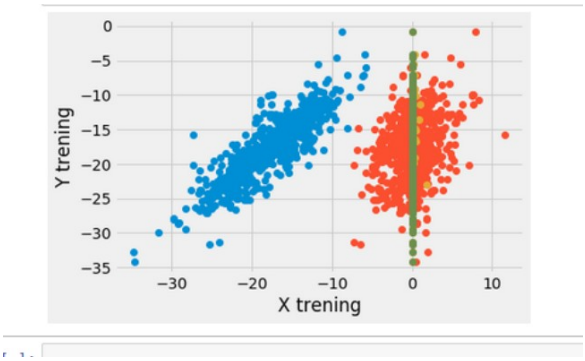
Slika 13: Rezultati za linearnu regresiju D1

Rezultati su zanimljivi, ovaj algoritam varira s izračunima tako da računa najoptimalnije regresijske koeficijente za najmanju MSE sumu, možemo ga koristiti za prediktiranje najboljeg linearnog regresijskog algoritama za riješavanje nekog sustava.



Slika 14: Regresijska Odstupanja D1

Odstupanja su nam gotovo minimalna, s tim da su naši rezultati dobro skalirani. Dobili smo jako visoku točnost prediktiranja i naš algoritam je jako točan.



Slika 15: Mapiranje za regresiju

Ovdje možemo vidjet postupak aproksimiranja trening skupa. Kao i na svim skupovima, možemo vidjet da su zeleni i žuti skupovi postavljeni na nuli. To je rezultat naša 2 stupca, 3. I 4. Koji su manje više stalno 0. U tom slučaju dosta varijacija možemo imati kao Iinaš D2 skup koji ima oko 30 značajki.

6. Clustering

Clustering metode su slične po ponašanju kao klasiifikacijske metode. Temelje se na istom konceptu grupacije, ali se dve metode ipak malo razlikuju. Clustering traži sličnosti nad nekim nagađinjama i featuresima pa onda grupira uzorke u svoje određene grupe, dok klasifikacijski algoritmi gledaju čistu sličnu atribua i karakteristika uzoraka [3]. **Clustering se koristi više za grupiranje i proučavanje sličnosti među neki uzorcima u istoj grupi**. Svako nagađanje koje naš stroj napravi bazira se direktno na algoritmu razmišljanja koje taj stroj nosi. Postoji nekoliko metoda razmišljanja koje opisujemo u nastavku.

6.1. Vrste clustering algoritma

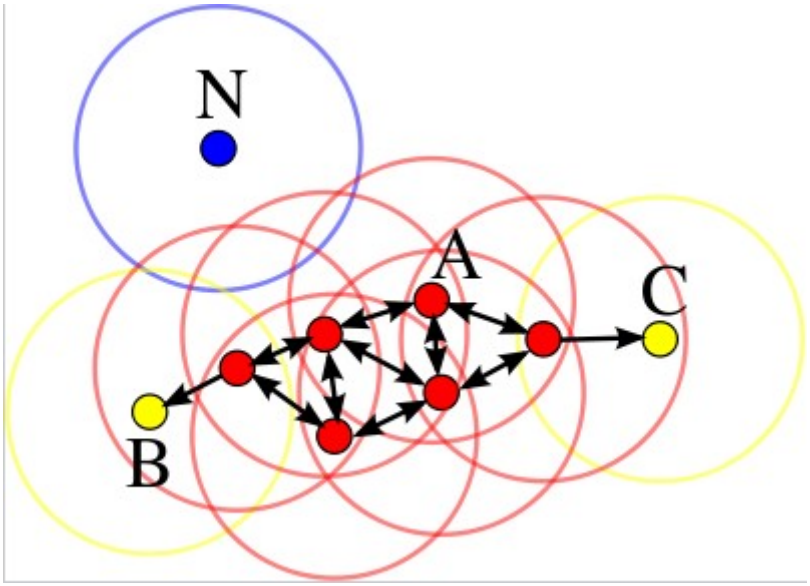
6.1.1. Density-based

Ovo razmišljanje je najednostavnije i najbrže za implementirati. Naime naš takozvani stroj „clustera" uzorku gusto populirane clustere ili nakupine podataka, biti će manje grupa i neke nakupine će se skupiti u jednu veću.

Jedan od najboljih density-based algoritama je algoritam iz 1996. zvan **Density-based spatial clustering of application with noise (DBSCAN)** [3].

6.1.2. DBSCAN

DBSCAN [3] funkcionira jednostavno: u nekom prostoru dodijelimo uzorke, a onda on grupira nakupine više uzoraka sa sličnim atributima i radi od njih nakupine i markira one uzorke koji nisu dio neke nakupine. Ovo nije klasifikacijski algoritam jer ne grupira podatke po neki klasifikacijima, nego samo pokazuje veze i odnose između međusobno različitih uzoraka. Cilj ovog algoritma nije da pokaže kako najbolje svrstati podatke u grupacije, nego kako podaci funkcioniraju na odnosu na druge.



Slika 16: Primjer ponašananja algoritma DBSCAN, preuzeto s Wikipedije[3]

Svaka točka **a** je core točka ako ima minimalni broj uzoraka **minpts** u njenoj udaljenosti ε. **A** je dostižna od **b** ako je u dosegu ε od **b**. **A** je dostižna iz neke druge točke ako se može doći do nje preko različitih radijusa drugih točaka. Sve ostale točke koje su nedostižne su „low points" [3].

```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N := RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) := Noise                               /* Label as Noise */
            continue
        }
        C := C + 1                                          /* next cluster label */
        label(P) := C                                       /* Label initial point */
        SeedSet S := N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point Q */
            if label(Q) = Noise then label(Q) := C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed (e.g., border point) */
            label(Q) := C                                   /* Label neighbor */
            Neighbors N := RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check (if Q is a core point) */
                S := S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

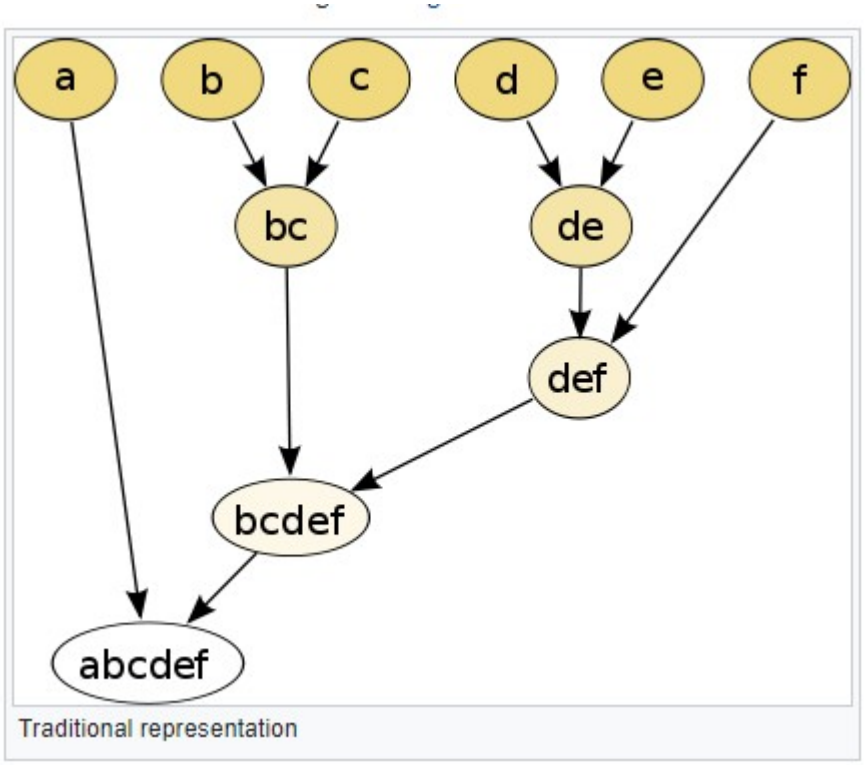
Slika 17: Orginalni DBSCAN kod[3]

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors N := empty list
    for each point P in database DB {                    /* Scan all points in the database */
        if distFunc(Q, P) ≤ eps then {                   /* Compute distance and check epsilon */
            N := N ∪ {P}                                 /* Add to result */
        }
    }
    return N
}
```

Slika 18: Range Query funkcija[3]

6.1.3. Hierachical–based

Ovaj način clusteranja se bazira na hijerahijskom pristupu gdje algoritam pokušava izgraditi hijerahijsko stablo iz danih uzoraka. Postoje dve vrste, a to su **Agglomerative** [5] **i Divisive** [5], a variraju se u izgradnji stabla, **agglomerative** ide iz listova do korijena a **Divisive** [5] suprotno



Slika 20: Sortirani ulaz[6]

Jedan od primjera **Agglomerative** [5] clusteringa, imamo listove koji se spajaju u čvorove pa tako se nastavlja do korijena. Međutim, clustera se po nekom uzorkovnom sistemu pa možemo imati listove koji idu direktno u korijen ako nemaju nikakve sličnosti s ostalim listovima.

6.1.4. Mjerenje clustering algoritma

Jedan od primjera Agglomerative clusteringa, imamo listove koji se spajaju u čvorove pa tako se nastavlja do korijena. Međutim, clustera se po nekom uzorkovnom sistemu pa to znači da možemo imati listove koji idu direktno u korijen ako nemaju nikakve sličnosti s ostalim listovima.

6.2. Kodovi

6.2.1. D1 kod za clustering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering

podaci = pd.read_csv('/home/adis/Desktop/RF/D1.csv')
podaci.head()

X = StandardScaler().fit_transform(podaci)

k_kof_8 = KMeans(n_clusters=8)
model = k_kof_8.fit(X)
y_hat_8 = k_kof_8.predict(X)
tagovi_8 = k_kof_8.labels_


metrics.silhouette_rezultat(X, tagovi_8, metric = 'euclidean')

metrics.calinski_harabasz_rezultat(X, tagovi_8)


sumaud = []
K = range(1,15)
for k in K:
    k_kof = KMeans(n_clusters=k)
    model = k_kof.fit(X)
    sumaud.append(k_kof.inertia_)

plt.plot(K, sumaud, 'ro-')
plt.xlabel('k')
```

```
plt.ylabel('Suma kvadriranih udaljenosti ')
plt.title('"Lakat" metoda za najbolji k')
plt.show()
```

6.2.2. D2 kod za clustering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering

podaci = pd.read_csv('/home/adis/Desktop/RF/d2.csv')
podaci.head()

X = StandardScaler().fit_transform(podaci)

k_kof_8 = KMeans(n_clusters=8)
model = k_kof_8.fit(X)
y_hat_8 = k_kof_8.predict(X)
tagovi_8 = k_kof_8.labels_

metrics.silhouette_rezultat(X, tagovi_8, metric = 'euclidean')

metrics.calinski_harabasz_rezultat(X, tagovi_8)

sumaud = []
K = range(1,15)
for k in K:
    k_kof = KMeans(n_clusters=k)
    model = k_kof.fit(X)
    sumaud.append(k_kof.inertia_)

plt.plot(K, sumaud, 'ro-')
plt.xlabel('k')
plt.ylabel('Suma kvadriranih udaljenosti ')
plt.title('"Lakat" metoda za najbolji k')
plt.show()
```

6.3. Objašnjenje koda za clustering

Osim poznatih biblioteka, ovdje se nalaze neke nove, a to su:

- **Kmeans** [8] –moduli iz sci-kita koji funkcionira slično kao DBSCAN, u smislu da kateogirzira clustere po nekom prostoru
- **AgglomerativeClustering** [8] – agglomerativni clustering koji je objašnjen iznad

skup prolazi kroz skaliranje, ali ovo je jedini skup koji ne sortiramo u testne i trening podatke. Ovdje uzimamo 8 clustera, možemo i više, ali ovo je broj koji je napovoljniji u testiranjima. Onda model, na atributima sortira clustere pomoću agglomerativnog clusterina. Ovdje je agglomerativni clustering algoritam razmišljanja, dok je **dbscan algoritam** koji sortira.

6.3.1. D1 Rezultati

```
In [5]: metrics.silhouette_score(X, tagovi_8, metric = 'euclidean')
Out[5]: 0.31419272268515225

In [6]: metrics.calinski_harabasz_score(X, tagovi_8)
Out[6]: 892.5797000952138
```

Slika 21: Silhouette rezultat za D1

Silhouette rezultat je 0.3, a može biti negdje od -1 do 1, što znači da je bolje nego prosječan. Nije idealan, ali je dosta dobar, i znači da su naši skupovi ipak imala neke zajedničke karakteristike.[8]

6.3.2. D2 Rezultat

```
In [5]: metrics.silhouette_score(X, tagovi_8, metric = 'euclidean')
Out[5]: 0.17679138341057876

In [6]: metrics.calinski_harabasz_score(X, tagovi_8)
Out[6]: 27.91787729235746
```

Slika 22: Silhouette rezultat za D2

D2 rezultat je znatno lošiji nego D1. Pretpostavljam da je razlog u strukturi podataka i da se događa overnamještanje modela. Za daljnju analizu D2 clustering algoritma, trebalo bi rekonstruirati tablicu i pogledati koje karakteristike uništavaju prosjek i nisu bitne za našu konstrukciju [8].

7. Zaključak

Kroz ovaj rad smo prošli neke od najboljih i najkorisnijih algoritama strojnog učenja koji postoje u današnje vrijeme. Možemo zaključiti da je tip svakog algoritma koji ćemo primijeniti ovisan o vrsti posla koji želimo obaviti. Dvije velike kategorizacije bi bile:

- određivanje već postojećih varijabli i grupacija novih varijabli u jednoj od klasifikacijskih grupa
- izračunavanje novih vrijednosti uzoraka na temelju starih uzoraka i korištenjem metode regresije (ili na neki drugi način) naći najbolju aproksimaciju koja nam odgovara

Iz proučavanja raznih karateristika, matematičkih pozadina i funkcionalnosti svakog od ovih algoritama, jasno se vidi da su umjetna inteligencija i strojno učenje grane informatike koje su se počele rapidno razvijati zbog njihovih širokih mogućnosti automatizacije postojećih procesa i optimizacije novih. Strojno učenje ima svoje mjesto u administrativnim poslovima, pogotovo kod klasifikacije dokumenata i sličnih poslova. Jedno od najraznovrsnijih područja gdje se strojno učenje koristi je sigurno marketing, gdje svoju primjenu ima za ciljano reklamiranje proizvoda odgovarajućoj publici. Neovisno o moralnim i etičkim pitanjima koja se vežu uz uporabu, jasno je da trenutno strojno učenje prolazi kroz produkcijsku razinu, ali već postoje specijalizacije algoritama koji se dodatno usavršavaju i sigurno će ih u budućnosti biti i više.

8. Literatura

[1] M. J. Gareth, W. Daniela, H. Trvor, and T. Robert, An Introduction to Statistical Learning: With Applications in R. 2013.

[2] N. Peter and J. R. Stuart, Artificial Intelligence: A Modern Approach. 1994.

[3] ”DBSCAN,” [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN.

[4] B. Yoshua, Deep Learning. MIT Press, 2015.

[5] A. Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow. 2017.

[6] ”Machine learning,” [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning.

[7] ”RxDock Documentation,” [Online]. Available: https://www.rxdock.org/documentation/devel/html/.

[8] ”Sci-kit Documentation,” [Online]. Available: https://scikit-learn.org/stable/.

[9] D. J. van D. AALT and M. J. J. B. Alexandre, ”Solvated docking: introducing water into the modelling of biomolecular complexes,” OXfor Press University, [Online]. Available: https://academic.oup.com/bioinformatics/article/22/19/2340/240908.

[10] H. Trevor, T. Robert, and H. F. Jerome, The Elements of Statistical Learning. 2001.

[11] B. Andriy, The Hundred-Page Machine Learning Book. 2019.