UNIVERZA V MARIBORU

FAKULTETA ZA NARAVOSLOVJE IN MATEMATIKO

Oddelek za matematiko in računalništvo

Krista Rizman Žalik

Algoritmi iskanja in odkrivanja znanja v podatkovnih bazah

Maribor, 2021

Predgovor

V današnji informacijski dobi je obdelava podatkov, strojno učenje in odkrivanje znanja v shranjenih podatkih v podatkovnih bazah osrednjega pomena.
Gradivo opisuje algoritme iskanja in algoritme strojnega učenja, ki so na voljo za odkrivanje znanja v relacijskih podatkih in omrežjih shranjenih v podatkovnih bazah.

Gradivo je namenjeno študentom programa predmetni učitelj, usmeritve izobraževalno računalništvo, pri predmetih Osnove podatkovnih baz in Osnove informacijskih sistemov. V pomoč pri zbranem gradivu so tudi obstoječi učbeniki, zbirke vaj in spletni viri.

Kazalo vsebine

Kazalo slik

1 Uvod

Relacijske podatkovne baze, ki danes še vedno prevladujejo, je definiral E. F. Codd daljnega leta 1970 [1]. Zaradi ogromnih količin podatkov z raznoliko strukturo so se razvile in uveljavile tudi NoSQL (angl. Not only SQL) podatkovne baze. Jedro podatkovne baze je sistem za upravljanje podatkovnih baz - SUPB (angl. Data Base Management System - DBMS). SUPB poleg številnih drugih funkcij optimizira izvedbo poizvedb. V tem gradivu opišemo algoritme za izvajanje poizvedb stikov podatkov iz več tabel.
Opišemo tudi algoritme, ki so vgrajeni tudi v relacijsko podatkovno bazo Oracle za strojno učenje in odkrivanje znanja v podatkih shranjenih v bazi.
Za odkrivanje znanja v povezanih podatkih v omrežjih opišemo algoritme, ki so na voljo tudi v bazi Neo4J. Neo4J je NoSQL podatkovna baza, ki temelji na grafu.

1.1 Struktura gradiva

Gradivo je razdeljeno na tri dele in dvanajst poglavij. Prvi del gradiva vsebuje dve poglavji in opiše algoritme za izvedbo stikov povpraševanj in delo z indeksi. Drugi del gradiva opiše algoritme za strojno učenje relacijskih podatkov. V tretjem delu gradiva so opisani algoritmi za odkrivanje znanja v omrežjih.

Uvodnemu poglavju sledi poglavje, ki opisuje algoritme iskanja in izvedbe stikov podatkov iz več tabel in uporabo indeksov v relacijskih podatkovnih bazah.
V tretjem poglavju spoznamo algoritme za izvajanje stikov poizvedb SQL v relacijskih podatkovnih bazah. Opišemo stik z vgnezdenimi zankami, stik z razprševanjem in stik z urejanjem in zlivanjem.

Sledijo najbolj razširjeni algoritmi za strojno učenje in odkrivanje znanja v podatkih.
V četrtem poglavju spoznamo algoritem Apriori, ki je namenjen iskanju povezovalnih pravil med elementi in iskanju pogostih množic elementov. V poglavjih od pet do osem spoznamo algoritem C4.5 za gradnjo odločitvenih dreves, algoritem gručenja podatkov k-means, metodo EM (angl. Expection Maximization) in metodo podpornih vektorjev. Deveto poglavje opiše Bayesovo klasifikacijo, ki določi verjetnost dogodka na podlagi predhodnega poznavanja pogojev povezanih z dogodkom. Deseto poglavje predstavi nevronske mreže, ki omogočajo strojno učenje. Nevronske mreže so zgrajene iz nevronov in posnemajo procesiranje možganov.

Sledijo algoritmi za odkrivanje znanja v podatkih, povezanih v omrežja, ki jih ponazorimo z grafi. Opisani razširjeni algoritmi so implementirani tudi v knjižnici Neo4J Data Science.
Enajsto poglavje opisuje algoritme ugotavljanja podobnosti vozlišč. Dvanajsto poglavje naniza merila in algoritme središčnosti, ki določajo pomembnost vozlišč v omrežjih. Opisan je algoritem izračuna središčnosti PageRank, ki so ga razvili pri podjetju Google leta 1996. Opisana so tudi naslednja merila središčnosti: središčnost lastnih vrednosti, središčnost vmestnosti, središčnost stopnje in središčnost bližine.
V trinajstem poglavju je opisana metoda Louvain vodilnega avtorja Blondel iz Univerze Louvain za odkrivanje skupnosti v velikih omrežjih.

Poglavje štirinajst opiše metodo širjenja oznak za iskanje skupnosti v omrežjih v skoraj linearnem času. Algoritmi iskanja najkrajše poti med dvema vozliščema so opisani v petnajstem poglavju.
V zadnjem, šestnajstem, poglavju opišemo merila za napoved povezanosti vozlišč.

A Algoritmi podatkovnih baz za izvedbo stikov in indeksov

2 Optimizacija povpraševanj in B-drevesni indeksi

Relacijske podatkovne baze je definiral E. F. Codd leta 1970 [1]. Tovrstne baze so sestavljene iz relacij – tabel z vrsticami in stolpci, ki so povezane s tujimi ključi. Jedro baze je sistem za upravljanje podatkovnih baz – SUPB. Del SUPB je orodje za optimizacijo poizvedb - optimizator. Optimizator poizvedb optimizira dostop do zahtevanih podatkov v poizvedbah in določa najučinkovitejši način za izvedbo stavkov SQL. SQL (angl. Structured Query Language) je standardni jezik, ki ga uporabljajo relacijske baze. Optimizator poizvedb združi, reorganizira in izvede poizvedbe, zapisane v SQL v poljubnem vrstnem redu. Optimizator izbere najoptimalnejši plan izvajanja poizvedbe in izdela drevo združevanja tabel. Pri tem upošteva razpoložljivo statistiko o podatkih (npr. prazni stolpci, velikosti tabel, …) in stroške vhodno/izhodnih operacij, centralno procesne enote in komunikacije. Podatki, zahtevani s poizvedbo, se poiščejo s popolnim pregledom tabel ali z uporabo indeksov. Optimizator izbira med različnmi metodami združevanja, ki je potrebno pri stikih dveh ali več tabel. Metode združevanja so vgnezdene zanke, razpršeni stiki (angl. hash join), stiki z urejanjem in zlivanjem, različni vrstni red združevanja tabel in možne transformacije. Optimizator poizvedb izbere plan izvajanja z najmanjšimi stroški, zato se imenuje optimizator, temelječ na stroških [2].

Indeksi omogočajo hitrejše iskanje zapisov. Indeks ustvari vnos za vsako vrednost, ki se pojavi v indeksiranih stolpcih. Relacijske podatkovne baze privzeto jo B-drevesne indekse, ki so primerni za primarne ključe. Ker imajo B-drevesni indeksi v zadnjem nivoju drevesa povezana vozlišča, so indeksi učinkoviti tudi za hitro urejanje po koloni indeksiranja [3].

2.1 B-drevesni indeksi

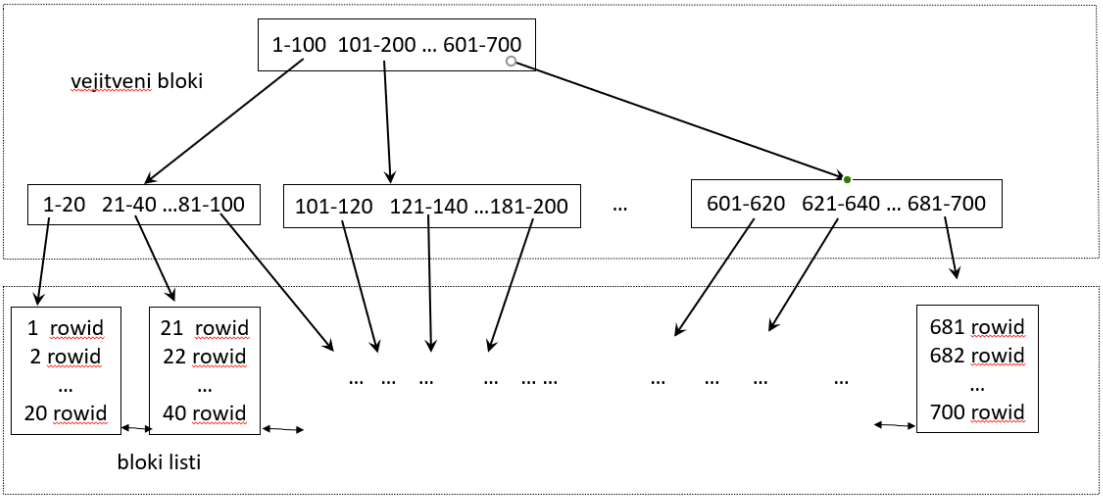
Indekse je uvedel Bayer McCreight v članku z naslovom »Organization and Maintenance of Large Ordered Indices« leta 1972 [3].

Poznamo več različnih B-dreves. Razlikujejo se po lastnostih, kot na primer, kje se hranijo dodatni podatki, ki pripadajo posameznim ključem in kakšne so omejitve stopnjam notranjih vozlišč. Tu si bomo pogledali osnovna B-drevesa in zaključili, da so vsi dodatni podatki, ki pripadajo posameznemu ključu, dostopni v številki vrstice *rowid*, ki je shranjena v listih B-drevesa poleg ključa [4]. B-drevo je kratica za uravnoteženo drevo (angl. balanced tree).

Za B-drevesne indekse velja [4]:
- B-drevesni indeks je najpogostejši indeks v podatkovnih bazah.

- B-drevo je samodejno uravnoteženo iskalno drevo za ogromno podatkov. B-drevesni indeksi s povezovanjem ključa z vrstico tabele s podatki zagotavljajo hitro iskanje podatkov, ki jih zahtevajo poizvedbe.
- B-drevesni indeks je urejen seznam vrednosti, razdeljen na območja zaporednih vrednosti v listih drevesa (slika 1).
- Primeren je za natančno ujemanje vrednosti ali iskanje znotraj območja vrednosti (kot je na primer 20 < starost < 30). Ker so listi drevesa dvosmerno povezani, je možno hitro urejanje podatkov.
- B-drevesa so namenjena obdelavi ogromne količine ključev, ki ne gredo v hitri pomnilnik. Ostala uravnotežena drevesa (kot rdeče-črno drevo ali drevo AVL) so običajno v hitrem pomnilniku. Ostala drevesa lahko tudi beremo z diska, samo počasneje delajo, ker je globina večja. Ogromno število ključev lahko shranimo na disk in jih beremo iz diska v obliki blokov. Čas branja iz diska je veliko večji kot je dostopni čas v hitrem pomnilniku.
- Operacije na drevesih (iskanje, vrivanje, brisanje, maksimum, minimum, itd.) zahtevajo *O(h)* dostopov do diska, kjer je *h* globina drevesa. Globino drevesa naredimo čim manjšo z vstavljanjem največjega možnega števila ključev v vozlišča B-drevesa.
- B-drevo je zgrajeno iz vejitvenih vozlišč in vozlišč listov. Vejitvena vozlišča vodijo iskanje željene vrednosti atributa. Vozlišča listi pa hranijo ključe in vrstice v tabeli (rowid), kjer so shranjeni ostali podatki poleg ključa (slika 1).
- Velikost vozlišč B-drevesa je enaka velikosti blokov na disku.

Primer B-drevesnega indeksa:



Slika 1: B-drevesni indeks

```
Algoritem za blok koren drevesa:
if (iskani_ključ >=1 and iskani ključ < =100)
       Go to Blok1
else if (iskani_ključ >=101 and iskani ključ <=200)
       Go to Blok2
else if (iskani_ključ >=201 and iskani ključ <=300)
       Go to Blok3
else if (iskani_ključ >=301 and iskani ključ <=400)
       Go to Blok4
else if (iskani_ključ >=401 and iskani ključ <=500)
```

```
       Go to Blok5
else if (iskani_ključ >=501 and iskani ključ <=600)
     Go to Blok6
else if (iskani_ključ >=601 and iskani ključ <=700)
      Go to Blok7
end
```

Za vsako vozlišče B-indeksnega drevesa velja (slika 1) [3]:
- Vsako vejitveno vozlišče B-drevesnega indeksa je polje več območij. Vsako območje je določeno s spodnjo in zgornjo mejo. Vsako območje hrani kazalec na naslednje vozlišče v B-drevesu indeksov.
- Vozlišča listi hranijo poleg indeksa še številko vrstice (rowid), kjer se v tabeli nahaja zapis s to vrednostjo indeksa.
- Vozlišča listi imajo kazalce na zaporedna vozlišča listov.

Za indeksno B-drevo velja [4]:
- Globina indeksa je število blokov, potrebnih za prehod iz korenskega vozlišča v vozlišče list.
- Indeksno B-drevo je uravnoteženo, ker vsa vozlišča listi samodejno ostanejo na isti globini. To pomeni, da iskanje katerega koli zapisa od koder koli v indeksu traja približno enako dolgo.
- Število ključev in kazalcev na druge bloke omejuje velikost blokov. Zato se hranijo minimalne predpone ključa za odločitev vejitev med dvema ključema.

2.1.1 Algoritem iskanja v B-drevesu

```
Vhod: k - ključ ki ga iščemoIzhod: rowid vrstice v tabeli s ključem k
Algoritem:
vozlišče = koren drevesa
while vozlišče.otrok != null
    poišči v vozlišču območje z iskanim ključem in povezavo na vozlišče
otroka
    // pomakni se za en nivo nižje v drevesu
    vozlišče = vozlišče otroka
end while

//Vozlišče je list drevesa, kjer bi lahko bil ključ k, ki ga iščemo.
//Najosnovnejše je zaporedno iskanje:
do
   čitaj zapis v vozlišču list;
    if (zapis.ključ = k)
      return zapis.rowid  //vrne številko vrstice v tabeli s ključem k
    else
        return null
while (ni konec zapisov v vozlišču and ključ_v_vozlišču != k)
```

2.1.2 Algoritem vstavljanja v B-drevo

```
Vhod:  zapis s ključem k Izhod: vstavljen ključ s številko vrstice rowid v
vozlišču list B-drevesa
Algoritem:
```

```
vozlišče = koren drevesa
while vozlišče.otrok != null
      // Dokler vozlišče ni list se
      // pomakni se za en nivo nižje v drevesu.
      vozliščeOče = vozlišče;
      vozlišče = vozlišče.otrok;
end while
if (vozlišče je polno)
    razdeliPoŠirini(vozliščeOče, vozlišče);
end if
//Vozlišče je list drevesa, kjer bi lahko bil ključ.
preberi zapis v vozlišču list
while (ni konec  zapisov v vozlišču and ključ_v_vozlišču < k) do
   preberi zapis v vozlišču list;
    if (naslednji _zapis.ključ <k )
   zapis = naslednji _zapis;
end do //  za zapis vrini nov zapis;
```

Če je vozlišče lista polno, ga je potrebno razdeliti na dva lista, polovico podatkov premakniti v nov blok in razdeliti območje v vozlišču očeta ter dodati novo povezavo na nov blok.

```
funkcija razdeliPoŠirini(vozliščeOče, vozlišče)
   pripravi blok(novBlok);
   prepiši v novBlok drugo polovico zapisov;
   kluč_delitve je vrednost ključa prvega prepisanega zapisa;
   v vozliščeOče najdi območje s ključem in prečitaj sedanjo k = končna meja
območja;
   popravi končno mejo na ključ_delitve -1;
   dodaj novo območje z mejama  ključ delitve in k ter kazalec = novBlok;
```

3 Stik

Stik dveh ali več tabel je opisan z SQL stavkom. Pogoj stika je opisan v `WHERE` delu stavka SQL (ne-ANSI SQL) ali delu `FROM ... JOIN` (ANSI SQL). ANSI SQL je SQL kot pa opiše standard Ameriškega nacionalnega inštituta za standarde (ANSI) iz leta 1986. Podatkovna baza primerja dve vrstici različnih tabel in upošteva pogoj stika. Če SQL poizvedba zajema podatke več tabel brez pogoja stika, baza naredi kartezični produkt in združi vsako vrstico prve tabele z vsako vrstico druge tabele [5].

3.1 Stik z vgnezdenimi zankami

Stik z vgnezdenimi zankami (angl. nested loops) je učinkovit [6]:
- ob stikih majhnih podmnožic podatkov,
- ob stikih velikih tabel, če je izbran način: FIRST_ROWS,
- če združujemo s pogojem, ki omogoča hitro doseganje notranje tabele.

Za stik z vgnezdenimi zankami velja [6]:
- Stik z vgnezdenimi zankami je enak vgnezdenima *for* zankama. Notranja zanka se izvede za vsako vrstico tabele zunanje zanke. V zunanji zanki optimizator poizvedb izbere tabelo, ki jo je potrebno pregledati v celoti (angl. full table scan), za notranjo, pa lahko uporabi indekse, ki hitro najdejo prave vrstice.
- Optimizator za odločitev, kako izvesti stik, ne uporablja velikosti celih tabel, ampak samo število vrstic, ki jih zahteva stik. Pogoj v stiku lahko, na primer, vključi le 10 vrstic iz tabele z dvema milijonoma vrstic.
- Če ima ena tabela v stiku samo eno vrstico npr. vrednost primarnega ključa (WHERE EMSO=1212212121212), izvede stik samo ena zunanja zanka.

3.1.1 Algoritem

Algoritem za pogoj stika tabela2.tuj_kluč = tabela1.primarni_kluč, kjer se notranja zanka izvede za vsako vrstico zunanje zanke:

```
FOR podatki1_vrstica IN (select * from tabela1) LOOP
  FOR podatki2_vrstica IN
    (select * from tabela2 where tabela2.tuj_kluč = tabela1.primarni_kluč)
LOOP
    OUTPUT  podatki1_vrstica+ podatki2_vrstica
  END LOOP
END LOOP
```

3.1.2 Primer

```
SELECT  fakultete.naziv, študenti.vpisna, študenti.priimek, študenti.ime
FROM    fakultete, študenti
WHERE   fakultete.fakulteta_id = študenti.fakulteta_id;
```

Tabele:

fakultete

id	okrajšava	Ime
10	FERI	
14	FGPA	
12	FKKT	
13	FS	
11	FF	

Za prvi id, ki je 10, poiščemo vse zapise v drugi tabeli študenti:

študenti

vpisna	fakultete_id	Priimek	Ime
1111111	10		
1111112	14		
1111113	12		
1111114	10		
1111115	13		
1111116	11		

In vstavimo v rezultirajočo tabelo:

naziv	Vpisna	priimek	Ime
FERI	1111111		
FERI	1111114		

Nato naredimo isto za drugi id v tabeli fakultete(14) in nato za tretji, ter ponavljamo do zadnjega.

```
SELECT  fakultete.naziv, študenti.priimek,  študenti.ime
FROM    fakultete, študenti
WHERE   fakultete.fakulteta_id = študenti.fakulteta_id;
```

Plan izvajanja za zgornjo poizvedbo:

```
-------------------------------------------------------------------------------
------------
| Id  | Operation                    | Name
-------------------------------------------------------------------------------
------------
|   0 | SELECT STATEMENT             |
|   2 |   NESTED LOOPS               |
|*  3 |    TABLE ACCESS FULL         | FAKULTETE
|*  4 |    INDEX RANGE SCAN          | ŠTUDENTI_FAKULTETA_IX |
|  5  |  TABLE ACCESS BY INDEX ROWID| ŠTUDENTI
```

Baza bere prvo tabelo fakultete vrstico za vrstico (angl. TABLE ACCESS FULL), da dobi vrednost polja fakulteta_id. Nato uporabi indeks ŠTUDENTI_FAKULTETA_IX , ki vrne številko vrstice (rowid) v tabeli študenti. Na koncu baza vrne podatke za številko vrstice v tabeli študenti [7].

3.2 Stiki z razprševanjem

Podatkovna baza uporabi stik z razprševanjem (angl. hash joins) za [8]:
- združitev večjih podatkovnih tabel,
- če gre za naraven stik, ki ima v pogoju enačaj,
- če je potrebno narediti stik na večjem delu podatkov manjše tabele in gre ta manjši nabor podatkov v hitri pomnilnik.

Razpršilni indeksi se pogosto uporabljajo kot primarni ključi ali enolični identifikatorji, ker shranjujejo vrednosti v indeksu in ne hranijo le kazalcev na zapise na disku kot drugi indeksi. To zagotavlja hitrejše iskanje in uporabo podatkov v indeksu.

Za stik z razprševanjem velja [8]:
- Optimizator izbere tabelo z manjšo množico podatkov v stiku za izgradnjo razpršene tabele po ključu.
- Optimizator uporabi deterministične funkcije razprševanja za določitev lokacije v razpredelnici, v katero naj se shrani vsaka vrstica.
- Podatkovna baza (SUPB) nato pregleda tabelo z večjim naborom podatkov in preišče razpršilno tabelo, da poišče vrstice, ki ustrezajo pogoju združevanja.

3.2.1 Algoritem

1. Baza prečita celotno manjšo podatkovno množico iz prve manjše tabele (full table scan) in izvede razpršeno funkcijo nad kolono stika ter tako zgradi razpršeno tabelo.

Psevdokod, kjer je funkcija razprševanje RAZPRŠI() in funkcija vstavljanja v razpršeno tabelo VRINI_RAZPRŠILNO_TABELO():

```
FOR majhna_tabela_vrstica IN
   (SELECT * FROM majhna_tabela)
LOOP
  št_vrste:= RAZPRŠI(majhna_tabela_vrstica.kolona_stika);
  VRINI_RAZPRŠILNO_TABELO(št_vrste, majhna_tabela_vrstica);
END LOOP
```

2. Baza uparja vrstice razpršene tabele z vrsticami iz druge tabele. Če so ustvarjeni indeksi po koloni stika za drugo tabelo jih uporabi, sicer pa čita celotno tabelo (full table scan).

Psevdokod:

```
FOR velika_tabela_vrsta IN
(SELECT * FROM velika_tabela)
LOOP
   št_vrste := RAZPRŠI(velika_tabela_vrsta.kolona_stika);
   mala_tabela_vrsta = POGLEJ_RAZPRŠILNO_TABELO(št_vrste,
velika_tabela_vrsta, kolona_stika);
   IF (majhna_tabela_vrsta != NULL)
   THEN
```

```
     OUTPUT mala_tabela_vrstica + velika_tabela_vrsta
   END IF;
END LOOP;

FUNCTION
      POGLEJ_RAZPRŠILNO_TABELO (št_vrste, velika_tabela_vrsta, kolona_stika)
//Pogleda v razpršeno tabelo v vrstico z izračunano številko vrstice
št_vrste.
IF (RAZPRŠILNA_TABELA[št_vrste] IS NULL)
      //v  razpršeni tabeli v vrsti s številko št_vrste ni zapisa
      vrstica = NULL;
   ELSE IF kolona != kolona_stika THEN
      Se kazalec pomika po povezanem seznamu,
      ki so se vsi preslikali v št_vrste dokler ne velja kolona=
kolona_stika;
      vrstica = RAZPRŠILNA_TABELA[št_vrste, kazalec]
   ELSE IF  (RAZPRŠILNA_TABELA[št_vrste].kolona == kolona_stika ) THEN
      vrstica= RAZPRŠILNA_TABELA[št_vrste]
    END IF
   RETURN(vrsta);
END
```

3.2.2 Primer razpršene tabele

Spodnja poizvedba zahteva razpršeno tabelo za fakulteta_id:

```
SELECT  fakultete.naziv, študenti.priimek, študenti.ime
FROM    fakultete, študenti
WHERE   fakultete.fakulteta_id = študenti.fakulteta_id;
```

1. Ker je tabela fakultete manjša od študentov, baza zgradi razpršeno tabelo za fakultete.

Prvih 5 vrstic tabela fakultete:

```
SQL> select * from fakultete where rownum < 6;
```

fakultete

id	okrajšava	ime
10	FERI	Fakulteta…
14	FGPA	Fakulteta…
12	FKKT	Fakulteta…
13	FS	Fakulteta…
11	FF	Fakulteta…

Predpostavimo, da ima razpršena tabela 5 vrstic in uporabimo preprosto zgoščevalno funkcijo F, ki vrne ostanek pri deljenju s 4:
F(10)=2, F(11)=3, F(12)=0, F(13)=1, F(14)= 2

Če se več različnih vrednosti razpršilne funkcije preslika v isti razpršilni ključ (angl. hash key) potem se vsi zapisi shranijo v isto vrsto razpršene tabele, pri čemer je prva vrednost vnesena v tabelo, do ostalih vrednosti pa se dostopa s pomočjo povezanega seznama. V našem primeru velja to za id 10 in 14.

Rowid	id	okrajšava	me		Povezan seznam		
0	12	FKKT	Fakulteta…				
1	13	FS	Fakulteta…				
2	10	FERI	Fakulteta…	→	14	FGPA	Fakulteta…
3	11	FF	Fakulteta…				

Plan izvajanja:

```
-----------------------------------------------------------------------------
| Id  | Operation            |  Name        | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------
|   0 |   SELECT STATEMENT   |              |
|*  1 |   HASH JOIN          |              |
|   2 |   TABLE ACCESS FULL  | FAKULTETE    |
|   3 |   TABLE ACCESS FULL  | ŠTUDENTI     |
-----------------------------------------------------------------------------
```

- Za tabelo fakultete naredi razpršilno funkcijo.
- Iz tabele študenti za vsako vrstico:
  - S pomočjo razpršilne funkcije F pridobi številko vrstice v razpršeni tabeli (rowid).
  - Uporabi isto razpršilno funkcijo F, kot za tabelo fakultete za zapis iz tabele študenti in dobi številko vrstice v razpršeni tabeli (rowid).
  - Prečitaj številko vrstice (rowid) iz razpršene tabele in zapis v povezanem seznamu s ključem stika ter vrni vsebino.

3.3 Stik z urejanjem in zlivanjem

Stik z urejanjem in zlivanjem (angl. sort merge join) uredi po velikosti dve tabeli ali več tabel, ki jih zahteva pogoj stika. Nato zlije po velikosti urejene tabele. Je pa urejanje časovno zamudna operacija. Če obstaja indeks z atributom stika, potem iz dvosmerno povezanih listov dobimo urejene vrednosti atributa po velikosti. Tako urejanje tabele po atributu stika ni potrebno. Stik z razpršeno tabelo je počasnejši od stika z urejanjem in zlivanjem, ko razpršena tabela ne gre cela v hitri pomnilnik. Optimizator izbere stik z urejanjem in zlivanjem in ne stik z razpršeno tabelo [9]:
- ko so podatki stika že urejeni ali
- ko je v stiku pogoj enakosti ali
- ko imamo velike množice podatkov.

Za razliko od stika z razprševanjem omogoča stik z urejanjem in zlivanjem delo z velikimi tabelami.

3.3.1 Algoritem za izvedbo naravnega stika z urejanjem in zlivanjem

Naravni stik je stik s pogojem enakosti. Izvedba naravnega stika z urejanjem in zlivanjem:

1. Urejanje: če tabeli, nad katerima se izvaja stik še nista urejeni, potem ju algoritem uredi po velikosti po atributu stika.

2. Zlivanje: bere vrstico iz obeh tabel. Če je kolona stika v drugi tabeli manjša bere naslednjo vrstico v drugi tabeli. Če pa je kolona stika v drugi tabeli manjša bere vrstico iz prve tabela. Ko pa sta koloni stika enaki, najde algoritem začetno vrstico v drugi tabeli.

```
READ  podatki1 SORT BY kolona_stika TO začasni_podatki1
READ  podatki2 SORT BY kolona_stika TO začasni_podatki2
READ podatki1_vrstica FROM začasni_podatki1
READ podatki2_vrstica FROM začasni_podatki2
WHILE NOT eof ON začasni_podatki1, začasni_podatki2
LOOP
    IF ( začasni_podatki1. kolona_stika = začasni_podatki2. kolona_stika )
      OUTPUT JOIN podatki1_vrstica, podatki2_vrstica;
      READ podatki2_vrstica FROM začasni_podatki2
    ELSIF ( začasni_podatki1. kolona_stika < začasni_podatki2. kolona_stika)
       READ podatki1_vrstica FROM začasni_podatki1
    ELSIF ( začasni_podatki1. kolona_stika > začasni_podatki2. kolona_stika)
       READ podatki2_vrstica FROM začasni_podatki2
END LOOP
```

3.3.2 Primer izvedbe naravnega stika z urejanjem in zlivanjem

```
SELECT  fakultete.naziv, študenti.vpisna, študenti. priimek, študenti.ime
FROM    fakultete, študenti
WHERE   fakultete.fakulteta_id = študenti.fakulteta_id;
```

Tabele s podatki, ki jih zahteva zgornja poizvedba:

Tabela fakultete:

id	naziv	ime
10	FERI	
14	FGPA	
12	FKKT	
13	FS	
11	FF	

Tabela študenti:

vpisna	fakultete_id	priimek	ime
1111111	10		
1111112	14		
1111113	12		
1111114	10		
1111115	13		
1111116	11		

Urejena tabela fakultete po atributu id:

Id	naziv	ime	ČASOVNO ZAPOREDJE ČITANJA ali PISANJA
10	FERI		1
11	FF		7
12	FKKT		10
13	FS		13
14	FGPA		16

Urejena tabela študenti po atributu id:

vpisna	fakultete_id	priimek	ČASOVNO ZAPOREDJE ČITANJA ali PISANJA
1111111	10		2
1111114	10		4
1111116	11		6

1111113	12		9
1111115	13		12
1111112	14		15

Rezultirajoča tabela poizvedbe:

Naziv	Vpisna	priimek	ime	ČASOVNO ZAPOREDJE ČITANJA ali PISANJA
FERI	1111111			3
FERI	1111114			5
FF	1111116			8
FKKT	1111113			11
FS	1111115			14
FGPA	1111112			17

3.4 Optimizator SQL oceni stroške združevanja

Optimizator SQL poizvedb oceni stroške posameznega načina združevanja in izbere najcenejšega.

Stroški za posamezno vrsto izvedbe stika so različni [9].

Stroški združevanja vgnezdenih zank so odvisni:
- od stroškov branja vsake izbrane vrstice zunanje tabele in
- hitrosti iskanja ustrezne vrstice notranje tabele.

Optimizator te stroške oceni s pomočjo statistike v slovarju podatkov.

Stroški metode stikanja z razprševanjem so odvisni:
- od stroškov izgradnje razpršene tabele na eni od vhodnih tabel spoja in
- hitrosti uporabe vrstic z druge tabele stika.

Stroški združevanja z metodo uredi in zlij so odvisni predvsem:
- od stroškov branja vseh podatkov v pomnilnik in
- od stroškov urejanja podatkov.

B Algoritmi za strojno učenje

Podatki, shranjeni v relacijskih podatkovnih bazah, in vgrajeni algoritmi za strojno učenje omogočajo iskanje še nepoznanih povezav med podatki in nudijo samodejno odkrivanje znanja, vzorcev in povezanosti med podatki.

Oracle® Machine Learning [11] nudi implementacije algoritmov za odkrivanje znanja, ki so vgrajeni v bazo (angl. in-database implementations) za naslednje različne naloge [10]:

- Regresija je tehnika napovedi. Algoritem za linearno in nelinearno regresijo je metoda podpornih vektorjev (angl. Support Vector Machine -SVM). Gaussovo jedro omogoča nelinearno regresijo, linearno jedro pa linearno regresijo.
- Odkrivanje anomalij in osamelcev. To so podatki, ki so daleč od večine ostalih podatkov. Za merjenje podobnosti uporabljamo različne metrike, npr. evklidsko razdaljo. Šum je enakomerno porazdeljen v celotnem podatkovnem prostoru ali delu podatkovnega prostora. Za odkrivanje anomalij lahko uporabimo metodo podpornih vektorjev (angl. Support Vector Machine - SVM).
- Tehnika asociacij poišče pravila v skupinah podatkov,ki se pogosto pojavljajo skupaj. Zelo razširjen algoritem je Apriori.
- Izbira lastnosti izdela nove atribute kot linearno kombinacijo obstoječih. Najbolj znana je metoda PCA (Principal Component Analysis), ki poišče nove sestavljene atribute, ki najbolje predstavljajo vse obstoječe atribute.
- Gručenje odkrije naravne gruče v podatkih. Algoritme gručenja lahko delimo v naslednje skupine [12]:

1. Modeli na osnovi bližine – razdalje med podatki. Na začetku vsak podatek predstavlja svojo gručo. Nato najbližje (najbolj podobne) podatke združimo in to iterativno ponavljamo dokler ne dobimo le ene gruče. To imenujemo aglomerativno hierarhično gručenje. Omogoča nam gradnjo in vizualizacijo hierarhij gruč, ki jih imenujemo dendogrami. Časovna in prostorska kompleksnost je velika O(n^3), zato je metoda primerna le za male podatkovne množice.

2. Modeli z voditelji - središči. Najbolj razširjena metoda z voditelji je k-means. Vsaka gruča ima središče in vsak podatek sodi v gručo z najbližjim središčem gruče. Na začetku središča gruč izberemo naključno. Nato v več iteracijah razporejamo podatke v gruče z najbližjim središčem in izračunamo nova središča gruč. To ponavljamo dokler se lega središč spreminja ali pa smo izvedli največje število iteracij, ki je določeno pred izvajanjem algoritma.

3. Porazdeljeni modeli. Porazdelitve določajo gruče.. Na primer Gausova porazdelitev lahko identificira simetrične gruče z gostim središčem. Multivariantno normalno porazdelitev uporablja algoritem EM (Expectation Maximization).

4. Modeli gruč, ki temeljijo na gostoti. Podatki, ki jih prikažemo v prostoru, tvorijo zgoščine in razredčine. Gostejši območja podatkov kot okolica tvorijo gruče. Podatki redkih območij so meje gruč, ki ločujejo gruče, ali pa so šum.

   O-Cluster je metoda na osnovi mreže (angl. grid) [13]. O-Cluster je hiter in prilagodljiv algoritem združevanja v gruče, ki je primeren za analizo velikih, visoko dimenzionalnih podatkov. Metoda identificira območja z visoko gostoto podatkov in loči gosta območja v gruče. Območja gostote odkrije z osmi vzporedne ortogonalne projekcije podatkov. Algoritem išče razcepljene skupine podatkov, ki spadajo v ločene gruče, ki se ne prekrivajo in so uravnotežene po velikosti. O-Cluster deluje rekurzivno in ustvarja hierarhijo. Število listov se lahko določi samodejno, lahko pa ga omejimo z določitvijo največjega števila gruč.

5. Gruče v omrežjih (grafih) imenujemo skupnosti. Skupnost predstavljajo vozlišča grafa-omrežja, ki so gosto povezana med seboj in redko z vozlišči drugih skupnosti. Razširjeni algoritmi so opisani v tretjem delu gradiva.

4 Iskanje pogostih podmnožic elementov in algoritem Apriori

Avtorja Agrawal in Srikant sta algoritem Apriori predlagala leta 1994 [14]. Algoritem je namenjen iskanju pogostih množic elementov (angl. frequent item set). Je najpogostejši algoritem za analizo nakupovalne košarice.
Algoritem Apriori najde posamezne pogoste elemente v podatkovni bazi in jih združi v pogoste pare, nato pogoste trojice, itd., dokler se ti dovolj pogosto pojavljajo v tabeli v podatkovni bazi [15].
Če posplošimo, dokler je število predmetov v novem naboru večje od 0 se:
1. Ustvari seznam kandidatov dolžine k (k-nabori).
2. S pregledom baze transakcij se za vsak k-nabor preveri njegova podpora.
3. Obdrži pogoste k-nabore in z njihovih kombinacij ustvari (k+1)-nabore.

Načelo Apriori trdi, da če je nabor pogost, so pogosti tudi vsi njegovi podnabori. Če nabor ni pogost, tudi nabori, ki ga vsebujejo, niso pogosti. Če označimo nabor vrednosti s *Fk* in pogostost nabora z *minimalna_podpora*, potem velja:

$\mathrm{Fk(X)} \geq \text{minimalna\_podpora} \Rightarrow \mathrm{Fk\ (Y)} \geq \text{minimalna\_podpora},\ \mathrm{Y} \subset \mathrm{X}$

Podpora pravilu določa pogostost pravila v celotnem številu transakcij. Zaupanje pravilu pa določa verjetnost, da se ob elementu1 v transakciji pojavi tudi element2.

$$podpora\ (element) = \frac{število\ transakcij\ v\ katerih\ se\ element\ pojavi}{celotno\ število\ transakcij}$$

$$zaupanje\ (element1 \rightarrow element2) = \frac{podpora(element1 \cup element2)}{podpora(element1)}$$

4.1.1 Algoritem Apriori

Naj bodo dovolj pogoste k-terice (nabori vrednosti) *Fk* za transakcijsko tabelo *Tabela*. *Kk* je množica kandidatov za k-terice, ki jih naredimo iz dovolj pogostih 1-teric shranjenih v Sk-1. Podpora pravil *podpora* je vhodni parameter.

```
Apriori (Tabela, podpora)
  S1 = SELECT izdelek from Tabela WHERE count(izdelek) > podpora;
  k = 2;
  while Sk-1 is not empty do
      //generira seznam kandidatov k-terice iz Sk-1 teric
      Kk = generirakterice (Sk-1,k)
      for transakcije t in Tabela
             for k-terica in Kk
                  //preverimo podporo k-tericam v transakcijah
                  if  k-terica je podmnožica t
                     Dt= Dt+k-terica in  Kk
                  end if
               end for
     end for
```

```
     for  kandidat kterica in Dt
              //štejemo k-terice v transakcijah
               število[kandidat kterica  ]++
      end for
      for c in Kk
           if  število[c] > podpora
                Fk = Fk+  c
                //  obdržimo pogoste k-terice
            end if
      end for
      k = k+1
    end while

generirakterice (Sk)
//  generira k-terice iz k-1 teric
     for all p ⊆ Sk, q ⊆ Sk where p1 = q1, p2 = q2, ..., pk-2 = qk-2 and
pk-1 < qk-1
        kandidat = p ∪ {qk-1}
          C = C + kandidat
     end for
     return C
```

4.1.2 Primer

Primer je izdelan v bazi MySQL.

```
set sql_safe_updates=0;
drop table racuni;
create table racuni (id int, ime varchar(24));

insert into racuni select 1,'kruh';
insert into racuni select 1,'mleko';
insert into racuni select 1,'papir';
insert into racuni select 1,'riž';
insert into racuni select 1,'sol';
insert into racuni select 2,'kruh';
insert into racuni select 2,'sol';
insert into racuni select 2,'jajca';
insert into racuni select 3,'solata';
insert into racuni select 3,'jajca';
insert into racuni select 4,'solata';
insert into racuni select 5,'jajca';
insert into racuni select 5,'kruh';
select * from racuni;
id         ime
1      kruh
1      mleko
1      papir
1      riz
1      sol
2      kruh
2      sol
2      jajca
3      solata
```

```
3      jajca
4      solata
5      jajca
5      kruh

-- 1 element
-- podpora = 2
-- samo elementov, ki ne zadostijo podpori = 2
drop view elementi1;

--posamični elementi  s podporo 2:
create or replace view elementi1 as
select id, ime  from racuni
where ime in
(select  b.ime from racuni b
group by b.ime
having count(*)>= 2);

select * from elementi1;
  1    kruh
1      sol
2      kruh
2      sol
2      jajca
3      solata
3      jajca
4      solata
5      jajca
5      kruh

-- pari elementov:
 create view  elementi2 as
 select  a.id, a.ime ime1, b.ime ime2
 from elementi1 a, elementi1  b
 where a.id = b.id and b.ime>a.ime

 select * from  elementi2;
id    ime1      ime2
 1     kruh   sol
2      jajca  kruh
2      kruh   sol
2      jajca  sol
3      jajca  solata
5      jajca  kruh
-- elementi2with support

 create view elementi2withsupport
 as select id,  ime1,  ime2 from elementi2
 where concat(ime1,' ',ime2) in
 (select  concat(c.ime1,' ',c.ime2) from elementi2 c
group by concat(c.ime1,' ',c.ime2)
having count(*)> 1);

 select * from  elementi2withsupport;
id     ime1       ime2
1      kruh   sol
```

```
2      jajca  kruh
2      kruh   sol
5      jajca  kruh

-- elementi za trojice
 create or replace view elemnti3 as
 (select id, ime from racuni
 where ime in (select ime1 from elementiza3));

 select * from elemnti3;

id          ime
1      kruh
1      sol
2      kruh
2      sol
2      jajca
3      jajca
5      jajca
5      kruh
 --trojice
 create view elementi3 as
select  a.id, a.ime as ime1, b.ime as ime2, c.ime as ime3
from  elemnti3 a, elemnti3 b, elemnti3 c
where a.id=b.id and a.id= c.id and b.ime>a.ime and c.ime>b.ime;

select * from elementi3;
id              ime1         ime2          ime3
2      jajca  kruh   sol


 create view elementi3withsupport
 as select id,  ime1,  ime2, ime3  from elementi3
 where concat(ime1,' ',ime2, ' ',ime3) in
 ( select  concat(ime1,' ',ime2,' ',ime3)  from elementi3
 group by concat(ime1,' ',ime2,' ',ime3)
 having count(*) >1)

select * from elementi3withsupport;
```

5 Gradnja odločitvenih dreves z algoritmom C4.5

Odločitvena drevesa omogočajo razvrščanje podatkov v razrede. Algoritem C4.5, ki zgradi odločitveno drevo, je predlagal Quinlan leta 1993 [16]. Algoritem se lahko uporablja za binarne in večrazredne probleme razvrščanja. Algoritem je hiter, tako med gradnjo, kot med uporabo.
C4.5 zgradi odločitveno drevo iz množice podatkov za učenje in uporablja koncept informacijske entropije. V vsakem vozlišču odločitveno drevo izbere atribut, ki najbolj učinkovito razdeli učno množico na podmnožice. Kriterij delitve je normaliziran informacijski prirastek (razlika entropije). Atribut učne množice z največjim informacijskim prirastkom je izbran za naslednje vozlišče v odločitvenem drevesu. V nadaljevanju algoritem podmnožice rekurzivno deli po še neuporabljenih atributih z največjim normaliziranim informacijskim prirastkom.

5.1 Algoritem C4.5

1. Za vsak atribut *atributA* algoritem poišče normalizirano informacijsko stopnjo prirastka za delitev drevesa na osnovi atributa *atributA*.
2. Atribut z največjo vrednostjo normalizirane informacijske stopnje prirastka označimo z *atributAnajboljši*.
3. Odločitvenemu drevesu doda vozlišče, ki deli po atributu *atributAnajboljši.*
4. Ponovi korake od 1 do 3 na še neuporabljenih atributih za razcep odločitvenega drevesa in doda ta vozlišča kot potomce vozlišča atributa atributAnajboljši.

5.2 Primer

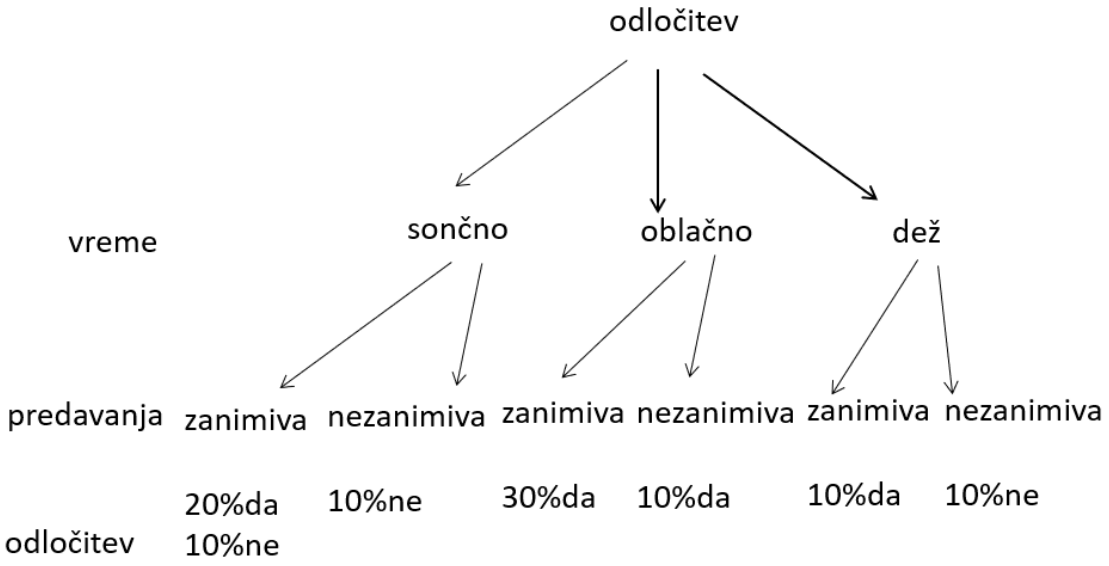
Deset dni spremljamo vreme in obiskanost predavanj. Odločitev »da« pomeni obisk predavanja in odločitev »ne« pomeni neudeležbo na predavanju. Izdelajmo odločitveno drevo.

Podatki za deset dni:

Dan	Vreme	Predavanja	odločitev
1	sončno	zanimiva	ne
2	sončno	nezanimiva	ne
3	oblačno	zanimiva	ja
4	oblačno	zanimiva	ja
5	oblačno	zanimiva	ja
6	dež	nezanimiva	ne
7	oblačno	nezanimiva	ja
8	sončno	zanimiva	ne
9	sončno	zanimiva	ja
10	dež	zanimiva	ja

```
Entropija(odločitev) = ∑ – p(I) . log2p(I) =
– p(Yes) . log2p(Yes) – p(No) . log2p(No) =
– (4/10) . log2(4/10) – (6/10) . log2(6/10) = 0.971

RazmerjePrirastka(A) = Prirastek(A) /
InfoDeljenja(A)

InfoDeljenja(A) = -∑ |Dj|/|D| x log2|Dj|/|D|

Prirastek(odločitev, predavanja) =  Entropija(odločitev) –
[ p(odločitev|predavanja=zanimiva) * Entropija(odločitev|predavanja=zanimiva)
+  odločitev|predavanja=nezanimiva) *
entropija(odločitev|predavanja=nezanimiva) ]

Atribut predavanja: zanimiva 7 , 23 nezanimiva, odločitev ne 4 krat in 6 krat ja


Entropija(odločitev|predavanja=nezanimiva) = – (2/3) . log2(2/3) – (1/3) .
   log2(1/3) = 0,918

Prirastek(odločitev, predavanja) = 0.971– (7/10)*(0.701) – (3/10)*(0.901)  = 0.21

InfoDeljenja(odločitev, predavanja) = -0,7*log(0,7) – (0,3)*log2(0,3) = 0.265

RazmerjePrirastka(odločitev, predavanja) = Prirastek(odločitev, predavanja) /
   InfoDeljenja(odločitev, predavanja) = 0.793

Atribut vreme: 4 sončno. 4 x oblačno in 2x dež, 4x ne in 6 x
ja

Entropija(odločitev|vreme=sončno) = – p(No) . log2p(No) – p(Yes) . log2p(Yes) =
– (3/4) . log2(3/4) – (1/4) . log2(1/4) = 0.811
Entropija(odločitev|vreme=oblačno) = – (0/4) . log2(0/4) – (1) . log2(1) = 0
Entropija(odločitev|vreme=dež) = – (1/2) . log2(1/2) – (1/2) . log2(1/2) = 1

Prirastek(odločitev, vreme) = 0.971– (4/10)*(0.811)– (4/10)*0 – (2/10)*(1)  =
0.447

InfoDeljenja(odločitev, vreme) = -0,4*log(0,4)-0,4*log(0,4)  – (0,2)*log2(0,2) =
0.458

RazmerjePrirastka(odločitev, vreme) = Prirastek(odločitev, vreme) /
   InfoDeljenja(odločitev, vreme) = 0.975
```

Vidimo, da je vrednost razmerja prirastka večji pri vremenu, zato bomo drevo najprej razdelili po vremenu in nato po drugem atributu Predavanja.
Slika 2 prikazuje dobljeno odločitveno drevo.



Slika 2: Odločitveno drevo za primer odločitve o obisku predavanj.

6 Algoritem gručenja k-means

K-means [17] je metoda za delitev podatkov v gruče podobnih podatkov. Število gruč *k* je vhodni parameter. K-means je razširjen algoritem gručenja avtorja MacQueen iz leta 1967.
Vsak podatek sodi v gručo z najbližjim središčem, ki lahko služi tudi kot prototip ali predstavnik gruče. K-means minimizira razdalje med središčem gruče in vseh točk v gruči.

6.1 Delovanje algoritma k-means

Algoritem k-means vhodno množico točk razdeli v *k* gruč, kjer je *k* vhodni parameter. Vhodni parameter je tudi največje število iteracij. V vsaki iteraciji k-means izbere voditelje - središča gruč in razporeja točke v gručo z najbližjim središčem. V osnovni verziji algoritma se središča v prvi iteraciji zberejo naključno. V vseh naslednjih iteracijah se središče vsake gruče izračuna kot povprečje vseh podatkov v gruči. Algoritem se konča, ko je doseženo največje število iteracij ali ko v iteraciji ni nobene spremembe položaja središč gruč.

Koraki algoritma k-means [17]:

```
Vhod: množica n točk
Vhodni parametri:
k - število gruč
i - največja vrednost iteracij, ki konča algoritem.
Rezultat: k gruč s središči. Središče gruče je središčna vrednost vseh
podatkov, ki pripadajo gruči. Točka pripada gruči, katere središče je točki
najbližje med vsemi k središči.

Izbira naključnih k središč gruč.
do
    i = i-1;
    1.Izračun razdalje od vsake točke do vseh k središč.
    2.Razporeditev vsake točke v gručo z najbližjim središčem.
    3.Izračun novih k središč, kot povprečna vrednost vseh točk gruče.
while i>0 or ni sprememb točk v gručah;
```

Časovna zahtevnost je približno linearna *O(n i k)*.

Kot omenjeno, je k-means je zelo razširjen algoritem gručenja. Ima pa kar nekaj slabosti. Težko je določiti število gruč *k*, ki je vhodni parameter. K središč lahko konvergira v lokalni ekstrem. Prav tako lahko oddaljene točke - osamelci (angl. outliers) in šum povzročijo, da algoritem ne najde realnih gruč.

6.2 Primera

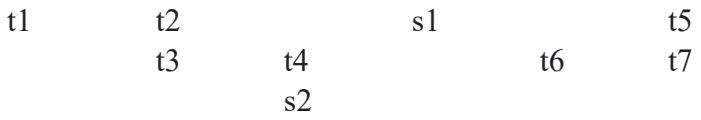
Primer 1:
Za dano množico podatkov *t* izračunajmo gruče s pomočjo algoritma k-means. Izdelajmo dve iteraciji. Število iskanih gruč je definirano s k=2. Začetni središči gruč sta s1 in s2. Uporabimo metriko razdalje Manhattan.

Metrika razdalje Manhattan:

$$m(A,B) = |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_n - b_n|$$

Središča:	x	Y
s1	4	1
s2	3	3

Podatki:	x	Y
t1	1	1
t2	2	1
t3	2	2
t4	3	2
t5	6	1
t6	5	1
t7	6	2



1. Razdalje podatkov do obeh središč:

Za izračun razdalj uporabimo razdaljo Manhattan.

	S1	s2
t1	3,00	4,00
t2	2,00	3,00
t3	3,00	2,00
t4	2,00	1,00
t5	2,00	5,00
t6	1,00	4,00
t7	3,00	4,00

2.Poiščemo vsaki točki t najbližje središče in dobimo 2 gruči:

C1={t1,t2 ,t5,t6,t7}

C2={t3,t4}

3. Izračun novih središč k1 in k2 obeh gruč:

Središča:	x	Y
s1	4	1,2
s2	2,5	2

2. izvajanje algoritma:

1. Razdalje točk do obeh središč:

Za izračun razdalj uporabimo razdaljo Manhattan.

	s1	s2
t1	3,20	2,50
t2	2,20	1,50
t3	2,80	0,50
t4	1,80	0,50
t5	2,20	4,50
t6	1,20	3,50
t7	2,80	3,50

2.Poiščemo vsaki točki t najbližje središče.
Točke ostanejo v istih dveh gručah:

C1={t1,t2, t3,t4}

C2={t5,t6,t7}

3. Izračun novih središč k1 in k2 obeh gruč:

Središča:	X	y
s1	2	1,5
s2	5,666667	1,333333

Primer 2:
Za dano množico točk iz prejšnjega primera izračunajmo gruče s pomočjo algoritma k-means. Uporabimo kvadrat evklidske razdalje. Število iskanih gruč je definirano s k=2. Začetni središči gruči sta k1 in k2.
Kvadrat evklidke razdalje v n-razsežnem prostoru izračunamo:
$d(A,B)^2 = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2$

Središča:	x	y
s1	4	1
s2	3	3

Točke:	x	y
t1	1	1
t2	2	1
t3	2	2
t4	3	2
t5	6	1
t6	5	1
t7	6	2

1. Razdalje točk do obeh središč:

Za izračun razdalj uporabimo kvadratno evklidsko razdaljo.

	s1	s2
t1	3,00	2,83

t2	2,00	2,24
t3	2,24	1,41
t4	1,41	1,00
t5	2,00	3,61
t6	1,00	2,83
t7	2,24	3,16

2.Poiščemo vsaki točki t najbližje središče in dobimo 2 gruči:

C1={t2 ,t5,t6,t7}

C2={t1,t3,t4}

3. Izračun novih središč k1 in k2 obeh gruč:

Središča:	x	y
s1	4,75	1,25
s2	2	1,666667

2. izvajanje algoritma:

1. Razdalje točk do obeh središč:

Za izračun razdalj uporabimo razdaljo Manhattan.

Točke:	s1	s2
t1	4,00	1,67
t2	3,00	0,67
t3	3,50	0,33
t4	2,50	1,33
t5	1,50	4,67
t6	0,50	3,67
t7	2,00	4,33

2.Poiščemo vsaki točki t najbližje središče.
Točke ostanejo v istih dveh gručah:

C1={t1,t2,t3,t4}

C2={t5,t6,t7}

3. Izračun novih središč k1 in k2 obeh gruč:

Središča:	x	Y
s1	2	1,5
s2	5,666667	1,333333

Vidimo, da dobimo isti središči gruč z uporabo razdalje Manhattan in kvadrata evklidske razdalje.

7 EM gručenje

EM algoritem (angl. Expectation Maximization) omogoča ocenitev mešanih modelov in je razširjena tehnika ocenjevanja verjetnosti gostote, ki se uporablja v različnih aplikacijah. Oracle uporablja EM za izvajanje distribucijskega algoritma združevanja v gruče (EM-clustering) [18].

EM algoritem je iterativna metoda. V koraku pričakovanja (angl. expectation step) se vrednosti parametrov nastavijo z ugibanjem. Vrednosti parametrov se uporabljajo za izračun verjetnosti trenutnega modela.
V kororaraku maksimizacije (angl maximization step) se ponovno izračunajo vrednosti parametrov, da se poveča natančnost modela.
Oba koraka se iterativno ponavljata do konvergence modela.

Koraki EM algoritma so podobni korakom K-means algoritma:
1. korak prireditve: Prireditev vsake točke najbližji gruči.
2. korak dodelave: Premik središča vsake gruče v središče vseh podatkov gruče.

EM gručenje izračuna delitev na gruče z uporabo verjetnosti gostote. Pri oceni gostote je cilj zgraditi funkcijo gostote, ki opiše porazdelitev. Področja z visoko gostoto podatkov v modelu ustrezajo vrhom gostote podatkov.

Gručenje, ki temelji na gostoti se razlikuje od gručenja, ki temelji na razdaljah in minimizira razdalje med vozlišči iste gruče ter maksimizira razdalje med vozlišči različnih gruč, kot na primer k-means.

Algoritem maksimizacije pričakovanj (EM) lahko uporabimo za ustvarjanje najboljše hipoteze za porazdelitvene parametre nekaterih multimodalnih podatkov. Gre za hipotezo največje verjetnosti - tista, ki poveča verjetnost, da podatki, ki jih gledamo, izhajajo iz k porazdelitev. Vsaka porazdelitev je določena s povprečjem m_K in varianco s^2za n podatkov.

Če uporabimo enojno modalno normalno distribucijo s povprečjem m in varianco s^2:

```
ocenjen m = ¬m~ = sum(xi)/n
ocenjen s2= s2~= sum(xi- m~)2/n
```

V več-modalni porazdelitvi moramo oceniti $h=[m_1, m_2, ..., m_K; s_1^2, s_2^2, ..., s_K^2]$. To je naloga EM algoritma.

Začnemo z neko začetno oceno za vsak m_K~ in s_K^2~. Ocene lahko dobimo z opazovanjem porazdelitev, iz prejšnjega znanja o domeni ali pa ugibamo. Nato vzamemo vsak podatek in poiščemo verjetnosti. Če imamo, na primer, dve porazdelitvi, poiščemo verjetnost, da je bila podatkovna točka xi, i = 1, ... n, vzeta iz N(m1 ~, sigma12 ~) in verjetnost, da je bila podatkovna točka xi, i = 1, ... n, vzeta iz N(m2 ~, sigma22 ~).

Za normalno funkcijo gostote izračunamo verjetnosti:
P(x_i pripada N(m_1~ , s_1^2~))=1/sqrt(2*pi* s_1^2~) * exp(-(x_i- m_1~)^2/(2*s_1^2~))
P(x_i pripada N(m_2~ , s_2^2~))=1/sqrt(2*pi* s_2^2~) * exp(-(x_i- m_2~)^2/(2*s_2^2~))

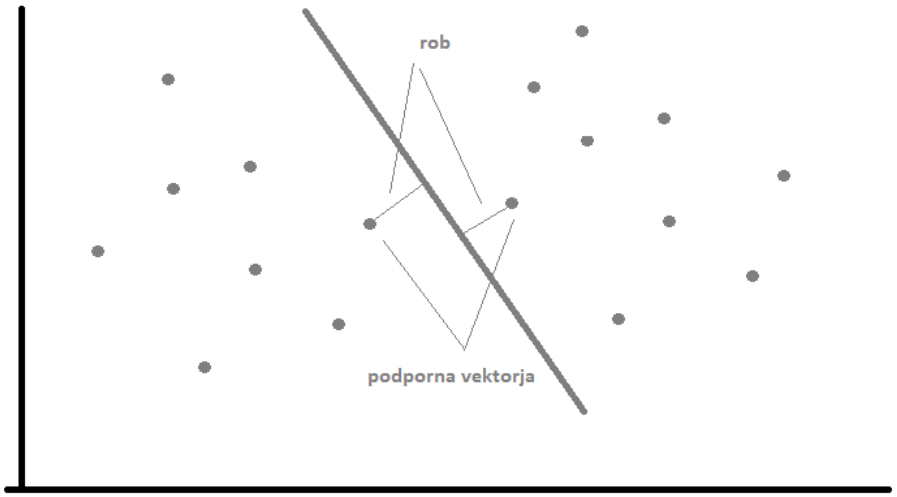
Upoštevati moramo tudi verjetnosti $\Omega k\sim$, da izberemo točko iz prve ali druge porazdelitve $N(m_1\sim, s_1^2\sim)$ ali $N(m_2\sim, s_2^2\sim)$ - razreda podatka Ck:

$$P(xi \in ck)) = \Omega k\sim P(xi \in N(m1\sim, s12\sim))/ \sum_{k=0}^{n} (\Omega k\sim P(xi \in N(m1\sim, s12\sim)))$$

Dva koraka algoritma EM sta:
1. E-korak: izračuna verjetnosti dodelitve vsake podatkovne točke nekemu razredu na podlagi trenutne hipoteze *h* za parametre distribucijskega razreda;
2. M-korak: posodobi hipotezo *h* za parametre distribucijskega razreda na podlagi novih dodelitev podatkov.
Slabosti metode so:
- Izbira ustreznih začetnih vrednosti parametrov lahko pomembno vpliva na kakovost rešitve.
- EM algoritem lahko konvergira do lokalnega maksimuma in ne do globalnega. Zato različne vrednosti začetnih parametrov vodijo do različnih parametrov in različne kakovosti modela.
- EM razvije model za velike gruče in ne odkrije podrobnega gručenja.
- EM komponente modela pogosto obravnava kot gruče, čeprav so naravne gruče različnih oblik in jih pogosto sestavlja več komponent modela. Oblika odkritih gruč je vnaprej določena z obliko funkcije gostote verjetnosti. Gaussova funkcija gostote lahko na primer identificira simetrične gruče z enim zgoščenim vrhom. Zelo gosta območja podatkov lahko EM algoritem obravnava kot samostojno gručo tako, da zgradi hierarhijo komponent, ki se lahko prekrivajo. Prekrivajoče komponente predstavljajo eno gručo. EM algoritem v hierarhiji komponent avtomatsko določi število gruč na najvišjem nivoju hierarhije. Za merilo prekrivanja uporablja Bhattacharya funkcijo razdalje [18].

8 Metoda podpornih vektorjev

Metoda podpornih vektorjev (angl. Support Vector Machine - SVM) [20] je algoritem učenja, ki se uporablja za klasifikacijo. SVM išče hiperravnino, ki najbolje deli učno podatkovno množico v dve množici.



Slika 3: Metoda podpornih vektorjev

Podporni vektorji so podatkovne točke, ki so najbližje hiperravnini. Če te točke odstranimo, se položaj razdelilne hiperravnine spremeni.
Hiperravnina je v podatkih z dvema lastnostima premica, ki linearno deli in klasificira množico podatkov.
Robna meja je razdalja med hiperravnino in najbližjo podatkovno točko iz obeh množic. Želimo hiperravnino z največjo možno robno mejo, kar daje večjo možnost pravilne razvrstitve novih podatkov.
Prednosti metode podpornih vektorjev:
- Natančnost.
- Učinkovitost na manjših in čistih (brez šuma) podatkovnih množicah.
- Učinkovitost povečamo z uporabo metode na podmnožici učnih podatkov.

Slabosti:
- Je neprimerna za večje podatkovne množice, ker je čas učenja SVM velik.
- Manj učinkovita za podatke s šumom in prekrivajočimi razredi.

SVM omogoča uporabnikom, da izberejo jedrno funkcijo (angl. kernel), ki najbolje ustreza podatkom, s katerimi delajo. Uporaba jedra za preslikavo naših podatkov v višje dimenzionalni prostor bo hiperravnini omogočila razdelitev podatkov.
Jedrna funkcija (angl. kernel) je funkcija za preoblikovanje podatkov. Primeri jedrnih funkcij sta linearna in polinomska.

9 Naivni Bayesov klasifikator

Naivni Bayesov klasifikator [21] omogoča hitro ocenitev razredov podatkov. Zahtevnost je linearna s številom napovedovalcev in vrstic. Bayesova klasifikacija opisuje verjetnost dogodka na podlagi predhodnega poznavanja pogojev, ki so lahko povezani z dogodkom.

Poglejmo si klasifikacijo na spodnjem primeru: Na fakulteti je 100 fizikov in slavistov. Kakšna je verjetnost da je ženska slavistka?

	moški	ženski	skupaj
slavist	20	25	45
fizik	50	5	55
skupaj	70	30	100

P(slavist | ženska) = P(slavist ∩ ženska) / P(ženska) = 25 / 30 = 0,83

X in Y sta dogodka, pogojna verjetnost ob znanem Y:

$$P(X \mid Y) = \frac{P(X \cap Y)}{P(Y)} P(X \mid Y) = \frac{P(X \cap Y)}{P(Y)}$$

P(Y | X) je neznanka, izračunamo jo iz učnih podatkov:

$$P(Y \mid X) = \frac{P(X \cap Y)}{P(X)} = \frac{P(X \mid Y) * P(Y)}{P(X)}$$

Bayesovo pravilo [20] je pravilo, kako iz znane verjetnosti *P(X | Y),* dobljene iz učne množice ugotovimo *P(Y | X).*
Za vse razrede *Y* (v spodnji enačbi ima spremenljivka Y vrednost *k*) izračunamo verjetnosti in največja verjetnost nam pove, v kateri razred sodi element.

$$P(Y = k \mid X) = \frac{P(X \cap Y = k)}{P(X)} = \frac{P(X \mid Y = k) * P(Y = k)}{P(X)}$$

Za računanje verjetnosti pri več pogojih lahko Bayesovo pravilo razširimo na Naivni Bayesov klasifikator. Klasifikator imenujemo naiven, ker predpostavlja, da so vse značilnosti med seboj neodvisne, kar skoraj nikoli ne drži. Za več spremenljivk - značilnosti *X1, X2,..., Xn* izračunamo verjetnosti [20]:

$$P(Y = k \mid X1, X2, \ldots Xn) = \frac{P(X1 \mid Y = k) * P(X\,2 \mid Y = k) * \ldots P(Xn \mid Y = k) * P(Y = k)}{P(X1) * P(X2) * \ldots P(Xn)}$$

9.1 Primer

Nekaj vrstic podatkovne množice za učenje:

	Mlajši	starejši	moški	ženska
slavist	1	0	0	1
fizik	0	0	1	0
slavist	0	1	1	0
fizik	0	0	0	1
…	…			

Število slavistov, fizikov in drugih ter skupne vrednosti:

	mlajši	starejši	moški	ženska	skupaj
Slavist	200	150	250	100	350
Fizik	100	50	100	50	150
Drugo	50	50	50	50	100
skupaj	350	250	400	200	600

```
k                                  P(Y =k)     P(Y =k)
slavist                            350/600 =   0,583333
fizik                              150/600 =   0,25
drugo                              100/600     0,166667

P(X1=mlajši)          =            350/600 =   0,583333
P(X3=moški)           =            400/600 =   0,666667
P(X1=mlajši | Y = slavist) =       200/350 =   0,571429
P(X1=moški | Y = slavist)  =       250/350 =   0,714286
P(X1=mlajši | Y = fizik)   =       100/150 =   0,666667
P(X1=moški | Y = fizik)    =       100/150 =   0,666667
P(X1=mlajši | Y = drugo)   =       50/100  =   0,5
P(X1=moški | Y = drugo)    =       150/100 =   0,5
```

Ali je mlajši moški najverjetneje fizik ali slavist ali kaj drugega? V računih spodaj vidimo, da je najverjetneje mlajši moški fizik.

$$P(Y = slavist|X1 = mlajši, X2 = moški) = \frac{P(mlajši|Y = slavist) * P(moški, Y = slavist) * P(Y = slavist)}{P(mlajši) * P(moški)}$$

$$= \frac{200/350 \ * \ 250/300 * 150/600}{350/600 \ * \ 350/600} \ = 0{,}2999$$

$$P(Y = fizik|X1 = mlajši, X2 = moški) = \frac{P(mlajši|Y = fizik) * P(moški, Y = sfizik) * P(Y = fizik)}{P(mlajši) * P(moški)}$$

$$= \frac{100/150 \ * \ 100/150 * 350/600}{350/600 \ * \ 350/600} \ = 0{,}762$$

$$P(Y = drugo|X1 = mlajši, X2 = moški) = \frac{P(mlajši|Y = drugo) * P(moški, Y = drugo) * P(Y = drugo)}{P(mlajši) * P(moški)}$$

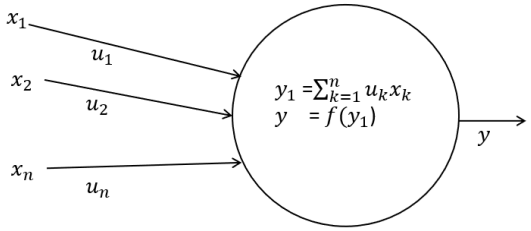
$$= \frac{50/100 \ * \ 100/150 * 100/600}{350/600 \ * \ 350/600} \ = 0{,}163$$

10 Nevronske mreže

Nevronska mreža je pomembna tehnika strojnega učenja in deluje po vzoru možganov [22]. Nevroni so gradniki živčevja, katerih glavna funkcija je proženje in prevajanje živčnih impulzov. Nevrone najdemo v možganih, hrbtenjači in živcih.
Nevronske mreže so sestavljene iz množice povezanih umetnih nevronov. Za nevronske mreže velja:
- Nevroni so osnovni gradniki umetnih nevronskih mrež. Nevron sestavlja več različno uteženih vhodnih povezav, aktivacijska funkcija in en izhod.
- Povezani umetni nevroni drug drugemu pošiljajo signale.
-Aktivacijska funkcija vsoto vhodnih signalov pretvori v izhod. Najpreprostejša je stopničasta aktivacijska funkcija, kjer se signal pojavi, ko je vsota vhodnih signalov dovolj velika. Vsak nevron ima tudi faktor pristranskosti (angl. bias), ki določa prag, pri katerem se bo pojavil izhodni signal.
- Povezave med nevroni imajo poljubno pomembnost, določeno z utežjo povezave.
- Uteži posameznih povezav se oblikujejo z učenjem. Učenje spreminja parametre nevronske mreže, dokler ni zmožna optimalno rešiti zastavljenega problema, ki ga opisuje množica učnih podatkov s poznanimi vhodi in izhodi. Učenje s pomočjo množice učnih podatkov imenujemo nadzorovano učenje (angl. supervised learning). Poznamo tudi druge oblike učenja nevronskih mrež, ki pa presegajo obseg tega gradiva, zato si jih ne bomo pobližje spoznali.
- Bistvo nevronskih mrež je v tem, da med učenjem same ugotovijo pravilo, ki povezuje izhodne podatke z vhodnimi.
- Nevronske mreže so sposobne učinkovito klasificirati podatke v razrede z iskanjem vzorcev podatkov - tradicionalnih transakcijskih podatkov, pa tudi slik, zvoka, pretočnih podatkov [22].

Prvi umetni nevron je bil Threshold Logic Unit (TLU) ali Linear Threshold Unit [23], ki sta jo predlagala Warren McCulloch in Walter Pitts leta 1943. Uporabljal je stopničasto funkcijo, ki proži nevron, ko je presežena pragovna vrednost $\boldsymbol{\mu}$.

Nevron
- sprejme vhode iz n virov,
- izračuna uteženo vsoto:
$y1 = x_1u_1 + x_2u_{12} \ldots + x_nu_n$,
- iz utežene vsote izračuna izhodno vrednost z uporabo aktivacijska funkcije
$y = f(y1)$ (slika 4),
- pošlje signal (*y) m* sosednim povezanim nevronom,
- natančno nevron matematično opišemo z enoplastnim modelom nevronske mreže:
$y = f(\sum_{i=1}^{n} x_iu_i + p)$,
kjer je f nevronska aktivacijska funkcija, *x* je vhodni vektor, y je izhodni vektor, u_i so uteži in *p* je faktor pristranskosti [24].

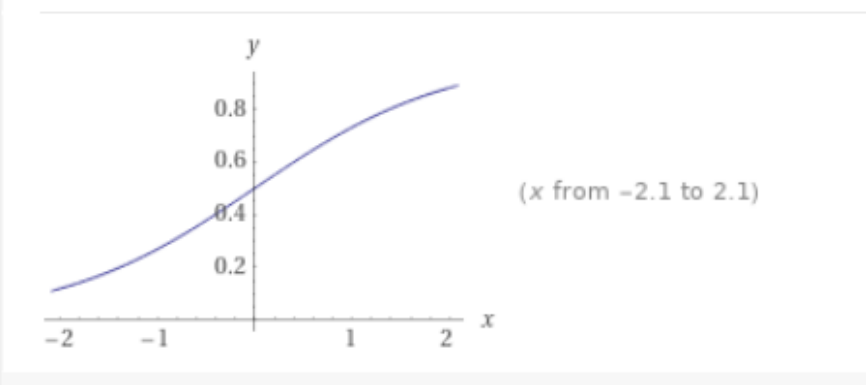


Slika 4: Nevron

Poleg stopničastih aktivacijskih funkcij poznamo še veliko drugih, kot so: logistična (sigmoidna) funkcija, hiperbolični tangens (tanh), arkus tangens (arctan), itd.

Sigmoidna funkcija je podana z enačbo:

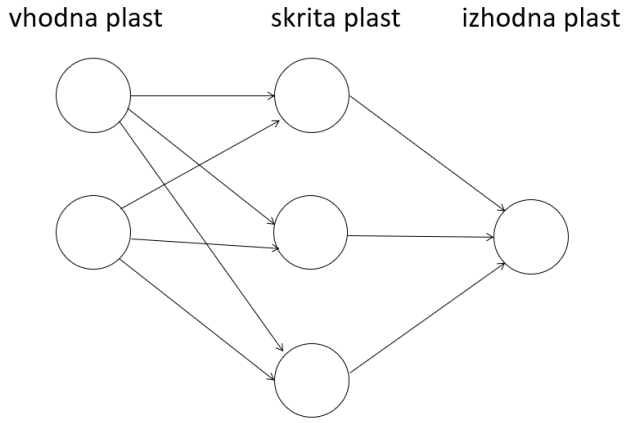
$$\sigma = \frac{1}{e^{-x} + 1}$$



Slika 5: Sigmoidna funkcija.

Najpreprostejša nevronska mreža se imenuje binarni perceptron. Mreža sestoji iz enega nevrona in se uporablja za nadzorovano učenje binarnih klasifikatorjev. Na izhodu da lahko vrednost 1 ali 0. Binarni perceptron si bomo podrobneje pogledali ob primeru v poglavju 10.1.

Kompleksnejše nevronske mreže so sestavljene iz več plasti. Prvo plast imenujemo vhodna, zadnjo plast izhodna, vmesne plasti pa so poimenovane skrite plasti. Slika 6 prikazuje nevronsko mrežo z eno skrito plastjo. Večina implementacij ima več skritih plasti, kar zahteva več računanja. Običajno so vsi nevroni ene plasti povezani z vsemi nevroni naslednje plasti.



Slika 6: Primer nevronske mreže z eno skrito plastjo.

Nevronske mreže za globoko učenje (angl. deep learning neural networks) [25] so nevronska omrežja, ki imajo več skritih plasti. Globina označuje število skritih plasti Primer nevronske mreže z eno skrito plastjo je na sliki 6.

Pri odkrivanju znanja v podatkih iz podatkovnih baz lahko nevronske mreže izvedejo klasifikacijo v več razredov ali regresijo za napoved neke številčne vrednosti.

Za učenje nevronske mreže velja:
- Najprej se uteži inicializirajo z nizom naključnih števil, enakomerno porazdeljenih znotraj območja, ki ga določi uporabnik (z nastavitvijo mej uteži.
- S širjenjem naprej (angl. forward propagation) mreža za vsak vhodni podatek izračuna vrednost izhoda.
- Nato sledi popravljanje uteži. Ker so za množico učnih podatkov znani pričakovani izhodi, se lahko izračunajo stopnje napak napovedanih izhodnih vrednosti. Stopnje napak se širijo po povezavah v obratni smeri (iz izhodne plasti v zadnjo skrito plast, itd.). Obdelava na vsakem nevronu povzroči prilagoditev uteži, da se stopnja napak zmanjša. Proces imenujemo širjenje nazaj (angl. backward propagation).
- Algoritem iterativno posodobi uteži v omrežju za vsak zapis iz testne množice, dokler mreža ni optimizirana.
- Premajhna testna množica lahko povzroči pretirano prilagajanje (angl. network overfitting), kjer se nevronska mreža nauči tudi nepomembnih lastnostih v vhodnih podatkih, kot so šum in podrobnosti, specifične za določene vhodno-izhodne pare učnih podatkov. Za pretirano prilagajanje je značilno, da nevronska mreža nad učnimi podatki deluje veliko bolje kot nad ostalimi podatki. Prekomerno prilagajanje najlažje odpravimo s povečanjem količine in raznolikosti učnih podatkov.

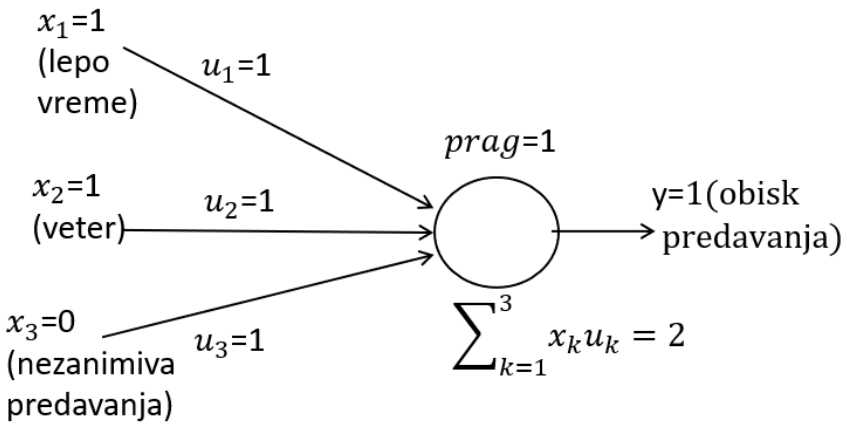
10.1 Primer perceptona

Preprost binarni nevron je perceptron. Slika 7 prikazuje učne podatke z atributi vreme, veter, predavanja. Izhod je odločitev: obisk predavanja ali izostanek na predavanjih. Perceptron lahko definiramo z enačbo:

$$y = P(\,u\,x - prag),$$

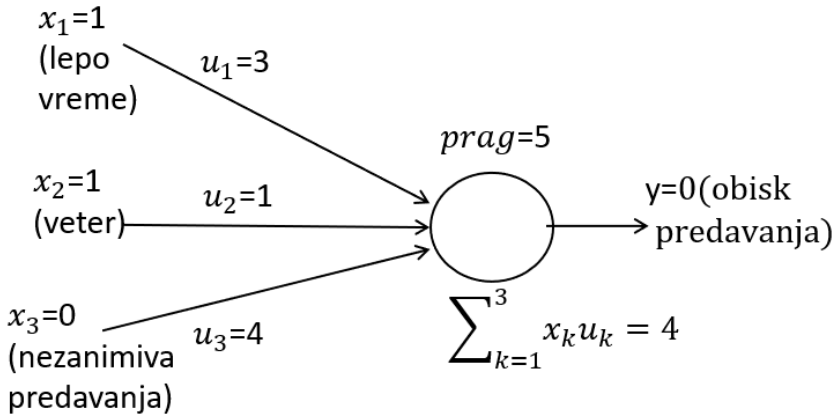
kjer je P pragovna aktivacijska funkcija, definirana z enačbo:

$$P(y1) = \begin{cases} 1; y1 \geq 0 \\ 0; \ sicer \end{cases}$$



Slika 7: Perceptron

Prag proženja v enačbi $y = P(u\,x - prag)$ opiše kako težko se percepton proži (slika 8).

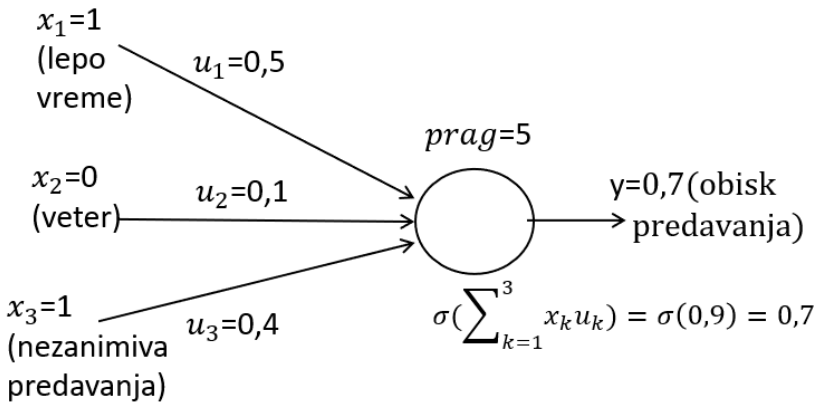


Slika 8: Perceptron z večjim pragom proženja.

10.2 Primer nevrona s sigmoidno aktivacijsko funkcijo

V tem primeru bomo pragovno aktivacijsko funkcijo zamenjali s sigmoidno, ki jo označimo z σ. Izhod nevrona ni ena izmed vrednosti 0 ali 1, ampak je realna vrednost iz območja med 0 in 1 (slika 9).

$$y = \sigma(u\,x - prag)$$

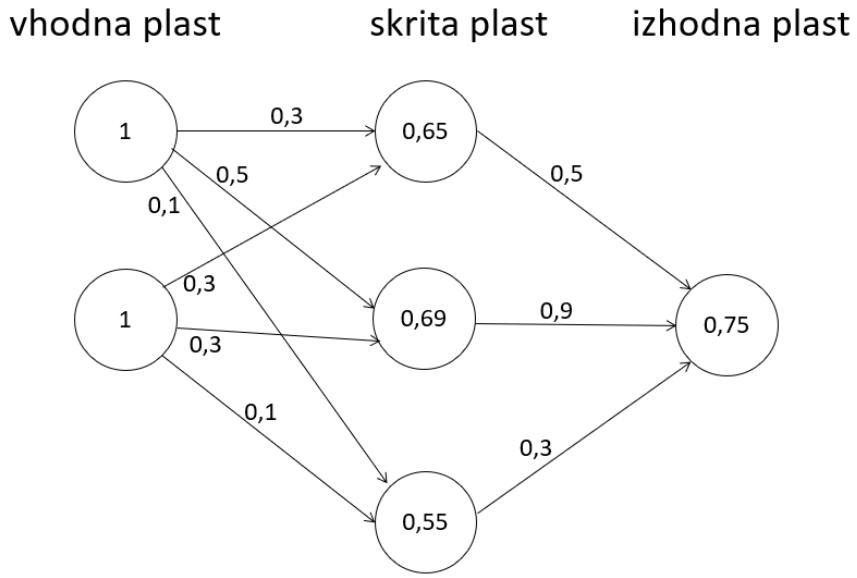


Slika 9: Nevron s sigmoidno aktivacijsko funkcijo

10.3 Preprosta dvoplastna umetna nevronska mreža

Uteži sinapsam izberemo naključno z Gaussovo porazdelitvijo. Inicializirane uteži so med 0 in 1, končne uteži so lahko večje od 1 (slika 10). Podobno kot pri prejšnjem primeru bomo v nevronih uporabili sigmoidno aktivacijsko funkcijo.

$h1 = \sigma(1 * 0,3 + 1 * 0,3) = \sigma(0,6) = 0,65$
$\mathrm{h2} = \sigma(1 * 0,5 + 1 * 0,3) = \sigma(0,8) = 0,69$
$h3 = \sigma(1 * 0,1 + 1 * 0,1) = \sigma(0,2) = 0,55$
$y = \sigma(0,65 * 0,5 + 0,69 * 0,9 + 0,55 * 0,3) = \sigma(1,111) = 0,75$



Slika 10: Preprosta dvoplastna nevronska mreža

Ker smo začetne uteži izbrali naključno, je rezultat mreže 0,75, pravi izhod bi moral biti 1. Sedaj s širjenjem nazaj popravimo uteži.

```
cilj=1
izračunano = 0,75
cilj – izračunano = 0,25
```

$\Delta vsote\ izhoda = \sigma(1{,}111) * 0{,}25 =$0,75*0,25=0,1875

$\Delta uteži_{u7}, u8, u9 = \frac{\Delta vsote\ izhoda}{rezultat\ skrite\ plasti} =$ $\frac{0{,}1875}{0{,}65\ \ 0{,}69\ \ 0{,}55}$ = 0,288 0,272 0,341

u7	0,5
u8	0,9
u9	0,3
u7'	0,788462
u8'	1,171739
u9'	0,640909

$\Delta vsote\ skriti = \frac{\Delta vsote_{izhoda}}{uteži_{skriti_{izhod}}} * [h1\ \ h2\ \ h3]$

$= 0{,}1875\ /[0{,}5\ 0{,}9\ 0{,}3]\ [0{,}65\ \ 0.69\ \ 0{,}55]$

```
= [ 0,375     0,208333   0,625] [0,65  0.69   0,55]
= [ 0,24375   0,14352 0,34375]
```

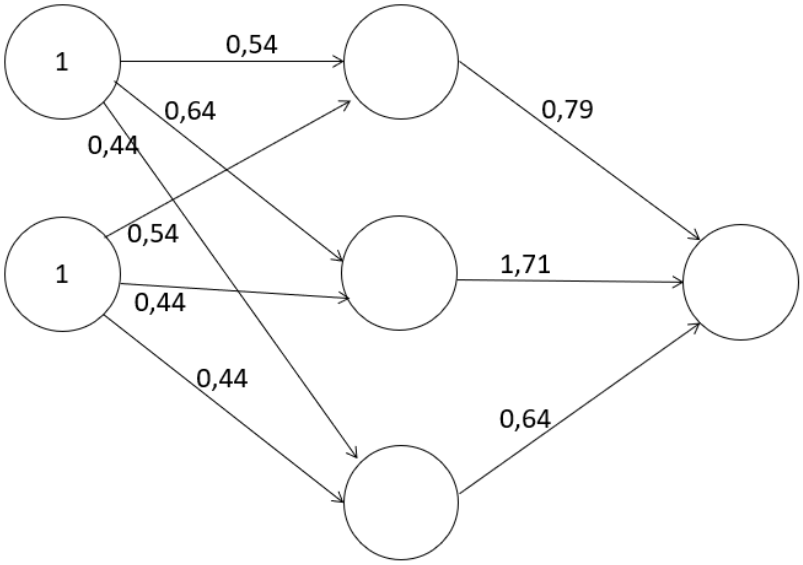
$\Delta uteži\_u1\_u6 =$ $\Delta vsote\ skriti$ / vhodni podatki = [0,24375 0,14352 0,34375] [1 1]

```
= [ 0,24375   0,14352   0,34375  0,24375   0,14352   0,34375]
```

u1=	0,3
u2=	0,5
u3=	0,1
u4=	0,3
u5=	0,3
u6=	0,1

u1'=	0,54
u2'=	0,64
u3'=	0,44
u4'=	0,54
u5'=	0,44
u6'=	0,44

Nevronska mreža s popravljenimi utežmi je prikazana na sliki 11.



Slika 11: Nevronska mreža s popravljenimi utežmi po enem učenju

Sedaj za ista vhodna podatka (1, 1) izračunamo izhodno vrednost z nevronsko mrežo (slika 11) mrežo s popravljenimi utežmi.

$h1 = \sigma(1 * 0{,}54 + 1 * 0{,}54) = \sigma(1{,}08) = 0{,}75$
$h2 = \sigma(1 * 0{,}64 + 1 * 0{,}44) = \sigma(1{,}08) = 0{,}75$
$h3 = \sigma(1 * 0{,}44 + 1 * 0{,}44) = \sigma(0{,}88) = 0{,}71$
$y = \sigma(0{,}79 * 0{,}75 + 1{,}71 * 0{,}75 + 0{,}64 * 0{,}71) = \sigma(2{,}33) = 0{,}91$

Izhodna vrednost je 0,91. Po eni iteraciji učenja je že izhodna vrednost 0,91 veliko bližja pravi vrednost 1, kot v primeru naključnih uteži (slika 10), kjer je bila izhodna vrednost 0,75.

C Algoritmi za strojno učenje mrežnih podatkov

Opisali bomo razširjene algoritme za delo nad grafi [26]. Implementirani so tudi v knjižnici Neo4J Graph Data Science Library. Neo4J je NoSQL podatkovna baza, ki temelji na podatkovnem modelu graf.

Na spletni strani je opisana funkcionalnost [27]: »Neo4J daje razvijalcem in podatkovnim znanstvenikom zanesljiva in napredna orodja za hitro izdelavo današnjih inteligentnih aplikacij in delovnih tokov strojnega učenja. Na voljo so kot popolnoma upravljana storitev v oblaku ali samostojna programska oprema.«

Graf $G = (V, E)$ je podatkovni model, ki sestoji iz množice vozlišč V, ki so med seboj povezana s povezavami E. Povezave so lahko usmerjene ali neusmerjene. Utežen graf je tisti graf, ki ima na vseh povezavah neko težo (npr. ceno, razdaljo). Ta nam pove koliko enot porabimo, da pridemo iz ene točke v drugo.

Neo4J Graph Data Science Library vključuje implementacije pogostih algoritmov nad grafi, ki opravljajo več funkcionalnosti [27]:

- Izračun podobnosti vozlišč: algoritmi izračunajo najbolj podobno vozlišče ali k-najbolj podobnih vozlišč posameznemu vozlišču grafa.
- Iskanje poti: algoritmi poiščejo najkrajšo pot ali ocenijo prisotnost ali kvaliteto poti.
- Izračun središčnosti: središčnost je merilo pomembnosti vozlišč. Izračun središčnosti tako omogoča razvrstitev vozlišč po pomembnosti.
- Odkrivanje gruč: algoritmi odkrivanja gruč razdelijo omrežje v več skupnosti s podobnimi vozlišči.
- Napoved povezav: algoritmi pare vozlišč na osnovi bližine.

11 Izračun podobnosti vozlišč

11.1 Podobnost vozlišč

Algoritem podobnosti vozlišč [28] primerja dve vozlišči glede na povezanost s sosednimi vozlišči. Vozlišči sta podobni, če imata večino skupnih sosedov. Algoritem izračuna medsebojno podobnost vozlišč z uporabo Jaccardove metrike. Ta izračuna razmerje med skupnimi sosedi in vsemi sosednimi vozlišči dveh vozlišč *A* in *B* z enačbo[29]:

$$J(A,B) = \frac{|A \cap B|}{|AUB|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

Algoritem podobnosti primerja vsako vozlišče z izhodnimi povezavami do drugih vozlišč. Za vsako vozlišče *n*, izberemo vsa sosednja vozlišča *N(n)* do katerih vodijo izhodne povezave tega vozlišča. Nepovezanih vozlišč ne rabimo preverjati.
Časovna kompleksnost algoritma je $O(n^2)$.

11.2 k-najbližjih sosedov

Metoda k-najbližjih sosedov [30] je opisana v članku: »Efficient k-nearest neighbor graph construction for generic similarity measures« prvega avtorja Wei Dong. Algoritem ne primerja vsakega vozlišča z vsemi drugimi vozlišči, ampak izbere vse možne sosede z upoštevanjem, da so sosedi sosedov najverjetneje najbližja vozlišča.

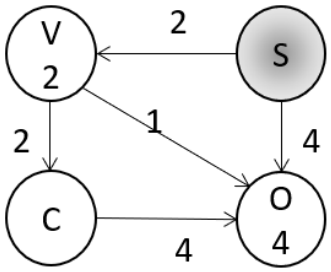
12 Algoritmi iskanja najkrajše poti med dvema vozliščema

Pot med vozliščema *S* in *C* je zaporedje povezav, ki vodijo iz vozlišča *S* v vozlišče *C*. Najkrajša pot med dvema vozliščema je tista, ki iz enega vozlišča pride v drugo tako, da je vsota uteži na povezavah po katerih gre pot najmanjša.

Edsger W. Dijkstra [49] je leta 1956 razvil algoritem, ki zgradi drevo najkrajših poti *D* iz poljubnega vozlišča grafa do vseh ostalih povezanih vozlišč. To je požrešna metoda s časovno zahtevnostjo $O(n^2)$, vendar lahko z uporabo kopice za iskanje naslednjega vozlišča zmanjšamo časovno zahtevnost na $O(m \log n)$, kjer je *m* število povezav in *n* število vozlišč v grafu.

Drevesu najkrajših poti lahko izdelamo usmerjen graf. Predpostavimo, da je vozlišče *V* že v drevesu. Povezavo $e = (V, O)$ lahko sprejmemo, če najkrajša pot od začetne točke *S* do točke *O* teče le po povezavah, ki so že v drevesu. Torej ne sme obstajati takšna točka *V*, ki ni v drevesu, in je pot iz izvora preko *V* do *O* krajša kot po povezavah, ki so že v drevesu. Da je pot od izvorne točke do točke *O* preko točk, ki so v drevesu krajša, pa mora veljati, da so vse točke, ki niso v drevesu kvečjemu bolj oddaljena od izvora, kot je točka *O*.

Primer:



Slika 12: Graf s štirimi vozlišči

Koraki najkrajših poti za graf na sliki 12, ki jih prikazuje tudi slika 13:
Najprej nastavimo razdalje *r* pri vseh vozliščih na ∞. V r_i hranimo trenutno najkrajšo pot od začetnega vozlišča, do vozlišča *i*. Nato si izberemo začetno vozlišče *S* na zgornjem grafu, razdaljo pri *S* nastavimo na 0 ($r_s = 0$).

a. Izberemo vozlišče *X*, ki še ni v *D* in ima najmanjšo trenutno razdaljo r_x. Razdalja r_x se več ne bo spreminjala in je predstavlja ceno najkrajše poti med *S* in *X*. Vozlišče X vstavimo v drevo najkrajših poti *D*.

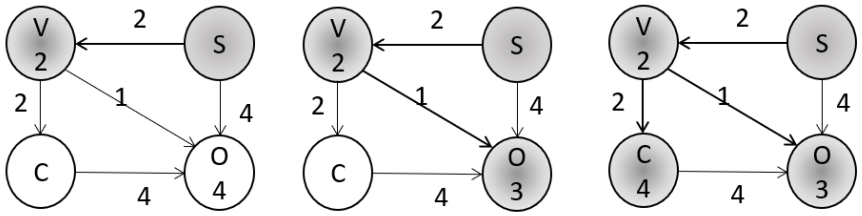
b. Nato preiščemo vse povezave, ki vodijo iz *X*. V primeru, da je pot iz začetne točke preko nove dodane točke do soseda Y krajša kot trenutno najkrajša r_y, potem vrednost r_y posodobimo, pri čemer moramo upoštevati dolžino povezave med *X* in *Y* $w_{x,y}$:

$$r_y = \min(r_y, r_x + w_{x,y})$$

c. Točki a in b ponavljamo, dokler niso vse povezave vstavljene v drevo najkrajših poti.

Delovanje algoritma je prikazano na sledečih slikah, pri čemer so vozlišča, vstavljena v *D* osenčena.

a) b) c)



Slika 13: Iteracije algoritma tvorbe drevesa najkrajših poti iz začetnega vozlišča S.

13 Algoritmi središčnosti

Različna merila središčnosti opisujejo različen tip pomembnosti vozlišča. Pomembno vozlišče je lahko vozlišče:
- ki ima največ sosedov ali
- ki je najbližje vsem vozliščem v grafu ali
- skozi katerega gre največ poti ali
- ki je povezano s čim več čim bolj pomembnimi vozlišči.

13.1 Središčnost PageRank

PageRank je merilo središčnosti, razvito v podjetju Google [31, 32]. Razvila sta ga ustanovitelja podjetja Larry Page and Sergey Brin. Uporablja ga njihov iskalnik, kjer imajo pomembnejše strani večji indeks PageRank. Vrednost PageRank uporabi optimizator spletnega iskanja.

PageRank analizira povezanost spletnih strani s pomočjo hiperpovezav. V HTML hiperpovezave opišemo z atributom *a href*. Primer:
Vadnica W3Schools.

Povezane spletne strani lahko prikažemo z grafom. Bolj povezane strani, ki so dosegljive iz več drugih spletnih strani, so bolj središčne in imajo večjo vrednost merila središčnosti PageRank.
PageRank torej meri pomembnost vozlišča v grafu na osnovi števila povezav sosednih vozlišč s tem vozliščem in pomembnosti teh sosednih vozlišč.

Avtorja sta PageRang matematično formulirala. V usmerjenem grafu je r_j merilo vozlišča (spletne strani) *j*. Če ima vozliče i d_i izhodnih povezav do drugih vozlišč, potem vsako vozlišče z izhodno povezavo iz vozlišča *i* dobi prirastek merila PageRank $\frac{r_i}{d_i}$. r_j za poljubno vozlišče *j* je torej vsota prirastkov vseh vhodnih povezav [31]:

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

Vsota vseh uvrstitev mora biti 1. Obojesmerna hiperpovezava privede do rekurzivne definicije. Rešimo jo s sistemom linearnih enačb, kar je lahko zamudna operacija za ogromno število spletnih strani. Zato so predlagani aproksimativni algoritmi.

13.1.1 Aproksimativni algoritem PageRank

Vrednosti PageRank lahko izračunamo z iterativnim reševanjem linearnih enačb, pri čemer na začetku inicializiramo vse uvrstitve na *1/n* za vseh *n* vozlišč usmerjenega grafa. Sistem linearnih enačb lahko izrazimo z matriko *D*, ki vsebuje elemente d_{ij} z vrednostjo $1/d_i$, (d_i je število izhodnih povezav), če obstaja povezava $i \to j$, sicer ima element vrednost 0 in dobimo:

$r^t = Dr^{t-1}$; $r_i^0 = \frac{1}{n}$

```
Vhod:
D, n, minPrirastek
Izhod: r
Algoritem:
t=0; // t je števec iteracij
```

$r_i^0 = \frac{1}{n}$

```
do
    t++;
```

$r^t = D\, r^{t-1}$

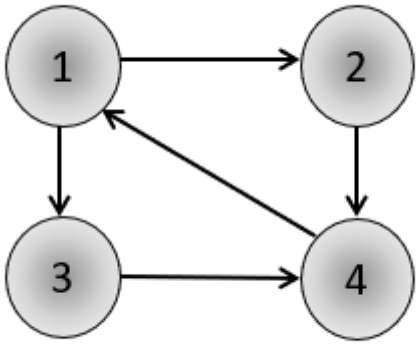
while $|r^t - r^{t-1}|$ > minPrirastek

Težave pri izračunu nastopijo, ko spletna stran nima izhodnih strani ali ob pojavu cikličnih hiperpovezav. Pri izračunu PageRanka se predvideva, da se strani brez odhodnih povezav povezujejo naključno z vsemi drugimi stranmi v zbirki. Ti naključni prehodi se dodajo vsem vozliščem v spletu. Preostalo verjetnostjo α običajno nastavimo na 0,85. Ocenjena je na podlagi pogostosti, da povprečen spletni uporabnik uporablja funkcijo zaznamkov (angl. bookmark) v svojem brskalniku. Torej je enačba naslednja [31]:

$$r_j = \alpha \sum_{i \to j} \frac{r_i}{d_i} + \frac{1-\alpha}{n}$$

13.1.2Primer

Izračunajmo središčnost PageRank za graf na sliki 14.



Slika 14: Preprost graf za izračun središčnosti

```
Sistem enačb:
r1=r4
r2=0,5 r1
r3=0.5 r1
```

r4= r2+r3

$$D=\begin{vmatrix} 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{vmatrix}$$

r^0 [0.25 0.25 0.25 0.25]

r1=0*0.25+0*0.25+0*0.25+1*0.25=0.25
r2= 0.5*0.25+0*0.25+0*0.25+0*0.25= 0,125
r3= 0.5*0.25+0*0.25+0*0.25+0*0.25= 0,125
r4= 0*0.25+1*0.25+1*0.25+1*0.25=0.5

r^1 [0.25 0.125 0.125 0.5]

r1= 0*0.25+0*0.125+0*0.125+1*0.5= 0,5
r2= 0.5*0.25+0*0.125+0*0.125+0*0.5= 0,125
r3= 0.5*0.25+0*0.25+0*0.25+0*0.5= 0,125
r4= 0*0.25+1*0.125+1*0.125+1*0.5=0.625

r^2 [0.5 0.125 0.125 0.625]

Vidimo, da je četrta stran najpomembnejša, saj ima največjo uvrstitev in vrednost PageRank. Nato ji sledi druga stran. Tretja in četrta stran sta najmanj pomembni. Imata obe isto najmanjšo vrednost središčnosti PageRank.

13.2 Središčnost lastnih vrednosti

Središčnost lastnih vrednosti (angl. Eigenvector Centrality) [33] meri vpliv vozlišč. Povezave iz vozlišč z večjo vrednostjo središčnosti bolj prispevajo k oceni središčnosti vozlišča kot povezave iz vozlišč z majhno središčnostjo. Visok rezultat lastnega vektorja pomeni, da je vozlišče povezano z mnogimi vozlišči, ki imajo sama visoke vrednosti središčnosti.

Središčnost lahko definiramo kot sorazmerno vsoti središčnosti sosednih vozlišč. Sorazmerno konstanto označimo z λ. Središčnost lastnih vrednosti vozlišča *i* označimo s *Ci*. Vhodno povezavo v vozlišče *i* iz vozlišča j označimo z a_{ji}. Potem velja [33]:

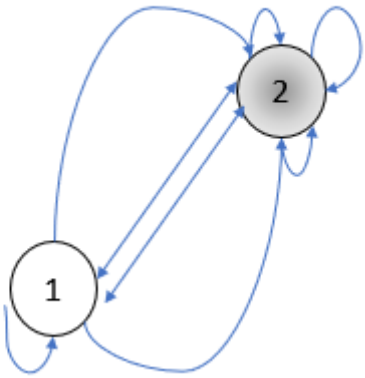
$$\lambda C_i = \sum_{j=1}^{N} a_{ji} C_i$$

Enačba zapisana v matrični obliki, kjer je *C* vektor središčnosti in je matrika *A,* matrika sosednosti ter je λ lastna vrednost [33]:

$$A^T C = \lambda C$$

Lastne vrednosti λ so številke, ki označujejo matriko. Lastne vrednosti v povezavi z lastnimi vektorji omogočajo izražanje matrike v poenostavljeni obliki, kar olajša izračune. Poenostavljena oblika je SDS^{-1}, pri čemer je S simetrična matrika, D pa diagonalna.

13.2.1 Primer središčnost lastnih vrednosti za omrežje z dvema vozliščema



Slika 15: Primer grafa z dvema vozliščema za izračun središčnosti lastnih vrednosti.

Izračunajmo središčnost vmestnosti za graf na siki 15.

$A = \begin{Bmatrix} 1 & 2 \\ 4 & 3 \end{Bmatrix}$

Lastne vrednosti matrike izračunamo tako, da izračunamo nule karakterističnega polinoma.
Karakteristični polinom dobimo, da odštejemo spremenljivko pomnoženo z identitetno matriko in izračunamo determinanto. Enačba za izračun determinante za matriko B velikosti 2x2:

$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix};$

$\det B = ad - bc$

$A = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix}$

$A - \lambda I = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1-\lambda & 2 \\ 4 & 3-\lambda \end{bmatrix}$

$\det(A - \lambda I) = (1-\lambda)(3-\lambda) - 2*4 = \lambda^2 - 4\lambda + 3 - 8 = \lambda^2 - 4\lambda - 5 = (a-5)*(a+1)$

`λ1=5; λ2=-1;` `sta lastni vrednosti.`

`Za λ1=5` $A - \lambda I = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} - 5 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -4 & 2 \\ 4 & -2 \end{bmatrix}$

$\begin{bmatrix} -4 & 2 \\ 4 & -2 \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \end{bmatrix} = 0$

$-4\,x1 + 2\,x2 = 0$

$4\,x1 - 2\,x2 = 0$

`x1=1, x2=2` $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ *je lastni vektor*

Vozlišče 2 ima večjo pomembnost kot vozlišče 1.

Za $\lambda 1 = -1$ $A - \lambda I = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} + 1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix}$

$\begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \end{bmatrix} = 0$

$2x1 + 2x2 = 0$
$4x1 + 4x2 = 0$

$x1=-1,\ x2=1$ $\quad \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ *je lastni vektor*

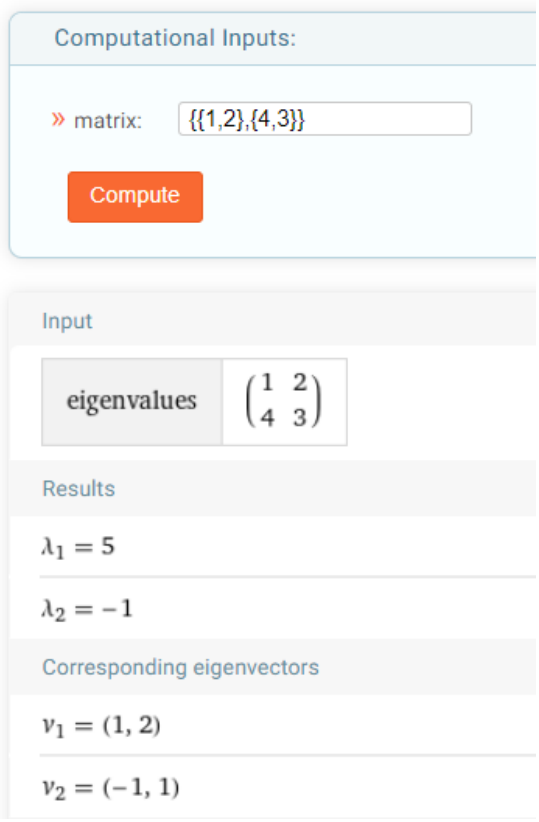
Lastne vrednosti (lambda) in lastne vektorje (v) lahko izračunamo s programom Octave (spletna različica na naslovu: https://octave-online.net/):

I

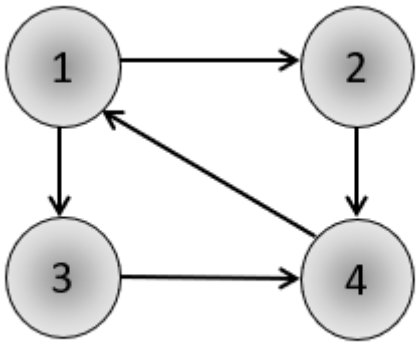
```
octave:5> A=[1 2;4 3]
A =
   1   2
   4   3
octave:6> [v,lambda]=eig(A)

v =
  -0.7071  -0.4472
   0.7071  -0.8944

lambda =
Diagonal Matrix
  -1   0
   0   5
```

Prosto programje Octave, ki ime ukaze, kompatibilne lastniški programski opremi MatLab, je za lastne vektorje izbral drugačne vrednosti od tistih, ki smo jih izbrali mi. Vendar sta razmerji v1,1 do v1,2 in v2,1 do v2,2 enaka naši rešitvi. Izbrani lastni vektorji sistema niso edinstveni, edinstveno pa je razmerje njihovih elementov. Vedno obstaja takšno število x s katerim pomnožimo dobljena vektorja, da dobimo naš vektor.

Lastne vrednosti (lambda) in lastne vektorje (v) lahko izračunamo tudi s spletnim programom WolframAlpha (na naslovu: https://www.wolframalpha.com/):



13.2.2 Primer središčnost lastnih vrednosti za omrežje s štirimi vozlišči



Slika 16: Graf s štirimi vozlišči

Izračunajmo središčnost vmestnosti za graf na siki 16.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Lastne vrednosti in lastni vektorji zračunani z Octave:

```
octave:1> A=[0 0 0 1;1 0 0 0; 1 0 0 0; 0 1 1 0]
A =

   0   0   0   1
   1   0   0   0
   1   0   0   0
   0   1   1   0

octave:2> [v,lambda]=eig(A)
v =

  -0.2549 - 0.4415i  -0.2549 + 0.4415i   0.5098 +      0i        0 +      0i
  -0.2023 + 0.3504i  -0.2023 - 0.3504i   0.4046 +      0i  -0.7071 +      0i
  -0.2023 + 0.3504i  -0.2023 - 0.3504i   0.4046 +      0i   0.7071 +      0i
   0.6423 +      0i   0.6423 -      0i   0.6423 +      0i        0 +      0i

lambda =

Diagonal Matrix

  -0.6300 + 1.0911i                  0                   0                   0
                  0  -0.6300 - 1.0911i                   0                   0
                  0                   0   1.2599 +      0i                   0
                  0                   0                   0        0 +      0i
```

Lastne vrednosti in lastni vektorji zračunani z Wolfram Alpha:

13.3 Središčnost vmestnosti

Intuitivno so opisovali središčnost vmestnosti (angl. Betweenness Centrality) že preden je Freeman [34] leta 1977 podal prvi formalni opis.

Središčnost vmestnosti najde vozlišče, ki je most med dvema delnima grafoma. Središčnost vmestnosti za podano vozlišče v izračunamo kot kvocient med vsemi potmi $p_{ij}(v)$ med vozlišči i in j, ki gredo skozi vozlišče v in tistimi potmi p_{ij}, ki ne gredo skozi v, za vse pare i in j iz množice vozlišč grafa, razen v ($V \backslash v$).

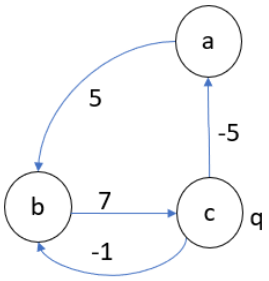
$$S(v) = \sum_{i \neq v \neq j} \frac{p_{ij}(v)}{p_{ij}}$$

Izračun zahteva O(n + m) prostora in časovno zahtevnost O(nm) in O(nm + n2 log n) za neuteženе grafe.

Algoritem za izračun je poimenovan po avtorju D. B. Johnsonu, ki ga je objavil leta 1977 [35].
Uporablja Bellman–Fordov algoritem za izračun najkrajših poti vozlišča do ostalih vozlišč v uteženem grafu, ki sta ga objavila Richard Bellman [36] in Lester Ford Jr [37] v letih 1956 in 1958. Leta 1959 je različico algoritma objavil tudi Edward F.

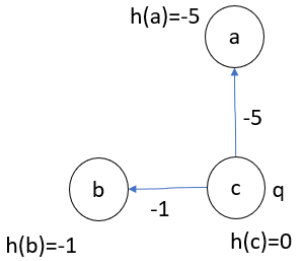
Moore [38] in zato včasih algoritem imenujemo Bellman–Ford–Moore. Je počasnejši kot algoritem Dijkstra ampak splošnejši in lahko obravnava povezave z negativnimi vrednostmi. Dijkstrov algoritem je praviloma hitrejši od Bellman–Fordovega algoritma, uteži

Algoritem za izračun središčnosti vmestnosti opišemo z naslednjimi koraki [35]:
1. V graf najprej dodamo novo vozlišče *q*, ki je do vseh ostalih vozlišč povezano s povezavami z utežjo 0.



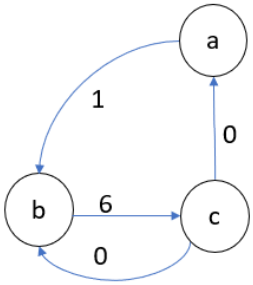
Slika 17: Graf z negativnimi povezavami

2. Nato uporabimo algoritem Bellman–Ford, ki začne v vozlišču *q* in za vsako vozlišče *v* poišče najmanjšo utež *h(v)* poti od *q* do *v*. V primeru negativnega cikla se algoritem konča.



Slika 18: Drevo najkrajših poti z uporabimo algoritma Bellman–Ford. Minimalne uteži poti od q do vsakega vozlišča *i* so označene s *h(i).*

3. Algoritem spremeni uteži povezavam grafa po izračunu Bellman–Ford algoritma: povezava iz vsakega vozlišča *u* do *v* z dolžino w(u,v) dobi dolžino w(u,v) + h(u) − h(v).

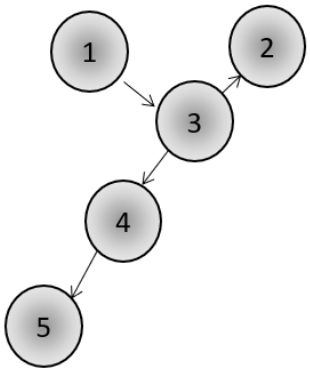


Slika 19: Ponovno utežen graf iz slike 15 s pozitivnimi povezavami

4. *q* umaknemo in algoritem Dijkstra poišče najkrajše poti od vsakega vozlišča *s* do ostalih vozlišč v ponovno uteženem grafu. V izvornem grafu izračunamo razdalje iz razdalje *D(u , v),* z dodajanjem *h(v) − h(u)* k razdalji dobljeni z algoritmom Dijkstra.

Ker je računanje natančne središčnosti vmestnosti zelo časovno zahtevno za velika omrežja, uporabimo aproksimativne algoritme, ki so hitrejši in še vedno dajo dobre rezultate. Brandes [39, 40] je ustvaril aproksimativni algoritem za neuteženе grafe s prostorsko zahtevnostjo *O(n+m)* in časovna zahtevnostjo *O(n * m)*, kjer je *n* število vozlišč in *m* število povezav.
RA-Brandesov algoritem (Randomized Approximate Brandes) je najbolj znan algoritem za izračun približnega rezultata za središčnost vmesnosti. Namesto da izračuna najkrajšo pot med vsakim parom vozlišč, torej iz vseh *n* začetnih vozlišč imenovanih tudi pivoti *s*, algoritem upošteva le podskupino vozlišč *S od* $\theta(\ log(n)/\ \varepsilon^2)$ pivotov. Dve skupni strategiji za izbiro podskupine vozlišč sta: naključna in na osnovi stopnje vozlišč. Najprej algoritem izračuna povprečno stopnjo vozlišč in nato obišče samo vozlišča z večjo stopnjo od povprečne. Nadaljnja optimizacija je, da omejimo globino najkrajših poti. Druga metoda za izbiro podskupine vozlišč je naključna.
Na redkih grafih je Brandesov algoritem bolj učinkovit in ima časovno zahtevnost $O(n^2 \log(n)+ n\ m)$. Na neuteženih grafih je časovna zahtevnost O(n m) [39].

13.3.1 Primer aproksimacije središčnosti vmestnosti



Slika 20: Graf s petimi vozlišči

Izračunajmo središčnost vmestnosti za graf na sliki 20.
Popoln algoritem:
število poti skozi 1: 0
število poti skozi 2: 0
število poti skozi 3: 3
število poti skozi 4: 2
število poti skozi 5: 0
število vseh poti: 3

Če upoštevamo najkrajše poti globine 1 se število poti zmanjša:
število poti skozi 1: 0
število poti skozi 2: 0

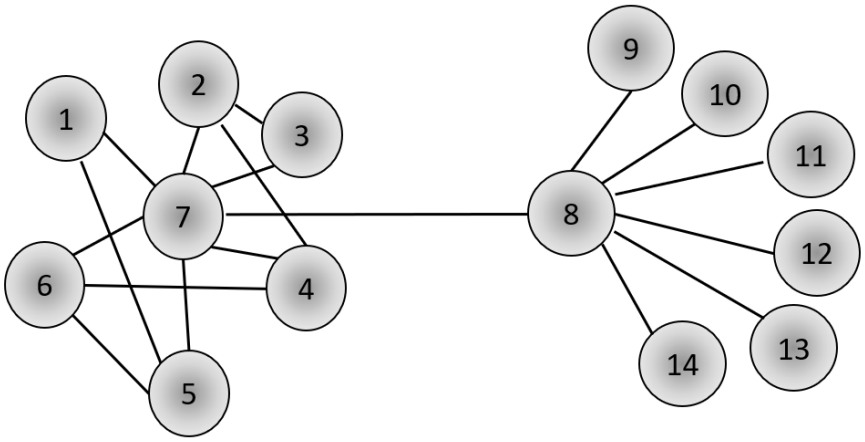
število poti skozi 3: 2
število poti skozi 4: 1
število poti skozi 5: 0
število vseh poti: 3

13.4 Središčnost stopnje

Središčnost stopnje (angl. Degree Centrality) [41] je najpreprostejše merilo središčnosti. Izračuna se iz števila sosedov vozlišča v grafu. Večje vrednosti povedo, da je vozlišče bolj središčno. Središčnost stopnje opisuje koliko povezav ima objekt.

13.4.1 Primer

Na spodnjem grafu (slika 21) imata vozlišči 7 in 8 isto središčnost stopnje, toda vozlišče 7 je vozlišče v središču omrežja, medtem ko je vozlišče 8 eno izmed perifernih vozlišč.



Slika 21: Graf vozlišč

13.5 Središčnost bližine

Bližino v grafu je definiral Bevales že leta 1950 [42] kot nasprotje oddaljenosti.
Središčnost bližine vozlišča (angl. closeness centrality) meri njegovo povprečno oddaljenost (inverzna razdalja) do vseh drugih vozlišč. Vozlišča z visoko oceno bližine imajo najkrajšo razdaljo do vseh drugih vozlišč.
Algoritem središčnosti bližine za vsako vozlišče izračuna vsoto razdalj do vseh drugih vozlišč na podlagi izračuna najkrajše poti med vsemi pari vozlišč. Središčnost bližine je obratna vrednost vsote:

$C(v) = \frac{1}{\sum_y d(v,y)}$,
kjer je *d(v, y)* je razdalja med vozliščema *v* in *y*.

Normalizacija omogoča primerjavo med središčnostjo bližine za grafe z različnim številom vozlišč. Normalizirano središčnost bližine dobimo z množenjem z *n-1,* za velike grafe lahko kar pomnožimo z *n*.

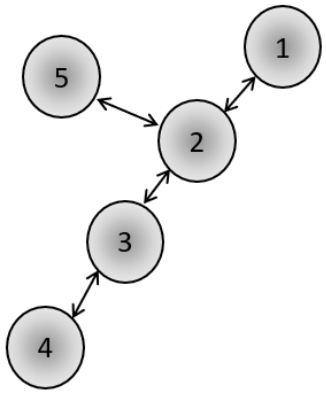
$$C(v) = \frac{n-1}{\sum_y d(vy)}$$

Središčnost bližine se uporablja za raziskovanje omrežij, kjer imajo posamezniki z veliko središčnostjo bližine ugoden položaj za nadzor in pridobivanje pomembnih informacij o celotnem omrežju.
Primer uporabe so različna omrežja z vodji, kot na primer omrežje upornikov v viru [43].
Središčnost bližine lahko uporabimo za odkrivanje žarišč, ki se lahko širijo tam, kjer se informacije razširijo po vseh najkrajših poteh hkrati. Tak primer je okužba z virusom, ki se širi preko socialnega omrežja [44].

Središčnost bližine omogoča odkrivanje vozlišč, ki lahko zelo učinkovito širijo informacije po grafu.

13.5.1 Primer

Za spodnji graf na sliki 22 izračunajmo pomembnost vozlišč z metriko središčnost bližine.



Slika 22: Usmerjen graf s petimi vozlišči.

n=5 // število vozlišč

Oddaljenost med vsakim parom vozlišč

Vozlišče	1	2	3	4	5	
1	0	1	2	3	2	
2	1	0	1	2	1	
3	2	1	0	1	2	
4	3	2	1	0	3	
5	2	1	2	3	0	
S	8	5	6	9	8	središčnost stolpca
(N-1)/S	0,5	0,8	0,6667	0,4444	0,5	normalizirana središčnost bližine

14 Metoda Louvain za odkrivanje skupnosti v omrežjih

Metoda Louvain se uporablja za odkrivanje skupnosti v velikih omrežjih. Razvil jo je Blondel iz University Louvain [45].
Je požrešna optimizacijska metoda. Zahtevnost metode je *O(n log n),* kjer je *n* število vozlišč v omrežju. Optimizira modularnost, to je vrednost med -0,5 (nemodularno združevanje v skupnosti) in 1 (popolnoma modularno združevanje v skupnosti), ki meri relativno gostoto povezav med vozlišči znotraj skupnosti glede na povezave med vozlišči skupnosti z vozlišči drugih skupnosti.

Modularnost izračunamo [45]:

$$Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right] \quad \delta(c_i c_j)$$

$A_{ij} - matrika\ sosednosti - uteži\ med\ vozlišči\ i\ in\ j$
$k_i - število\ sosedov -$ vsota $uteži\ povezav\ vozlišči\ i$
$m - število\ povezav -$ vsota $uteži\ povezav$
$c_i, c_j - skupnost\ vozlišča\ i\ in\ j$
$\delta(c_i c_j)$ – funkcija Cronecker delta; $\delta(c_i c_j) = 1\ če\ je\ c_i = c_j, sicer\ 0.$

14.1.1 Dva koraka metode Louvian

Prvi korak:
1. Na začetku je vsako vozlišče svoja skupina.
2. Za vsako vozlišče preverimo sosede (*j*) in ocenimo povečanje modularnosti, če se vozlišče *i* preseli v j-to skupnost.
3. Vozlišče *i* premaknemo v skupnost, ki ima največji pozitivni prirastek modularnosti.
4. Algoritem izvede točki 2 in 3 za vsa vozlišča. Ustavi se, ko ni več mogoče doseči povečanja modularnosti.

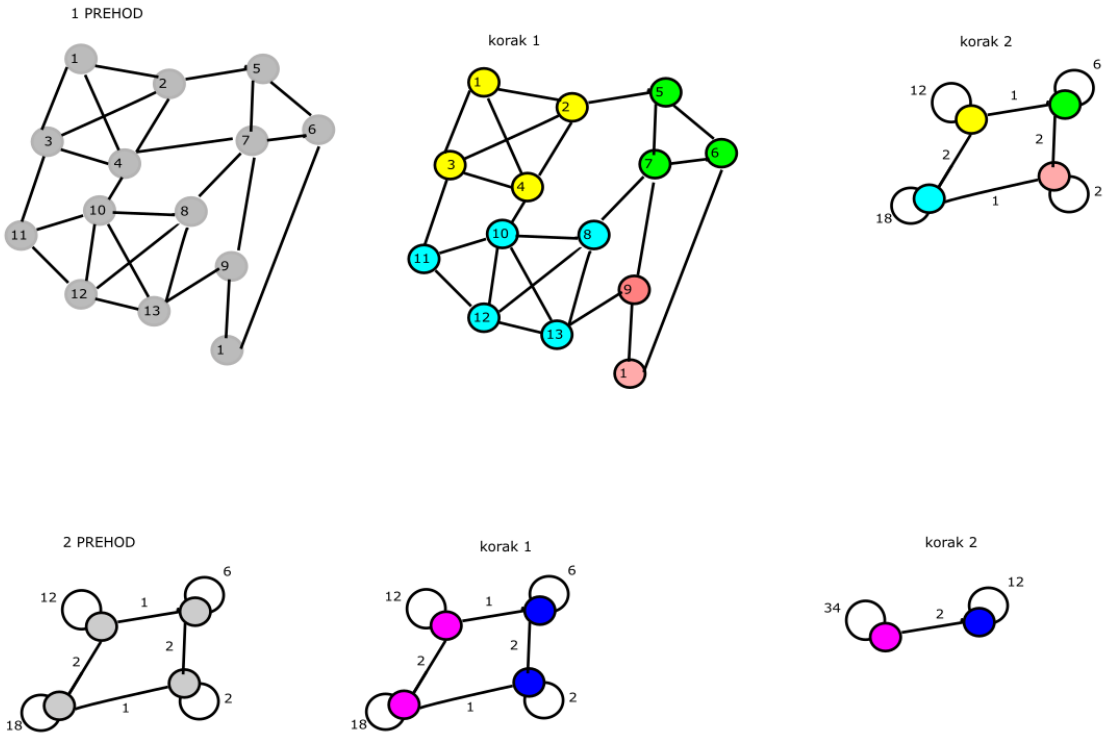
Prirast modularnosti za vstavljanje vozlišča *i* v skupnost *j* izračunamo po enačbi [47]:

$$\Delta Q = \left[\frac{\sum_{in} + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

$\sum_{in}$ vsota povezav-uteži povezav znotraj skupnosti ,
$\sum_{tot}$ vsota vseh povezav-uteži povezav vozlišč skupnosti

Drugi korak:
Algoritem tvori omrežje vozlišč, ki so bile v prvem koraku oblikovane v skupnosti. Povezave med vozlišči dobijo vsoto uteži med vozlišči ustreznih skupnosti. Povezave vozlišč samih s seboj dobijo uteži kot vsoto uteži povezav med vozlišči znotraj skupnosti.

Algoritem ponavlja oba koraka dokler ni nobenega premika vozlišča v drugo skupnost, kar pomeni, da je dosežena največja modularnost [47] (slika 23).



Slika 23: Primer prehodov in korakom metode Louvian

15 Širjenje oznak za odkrivanje skupnosti

Algoritem širjenja oznak (angl. Label Propagation) [48] odkrije skupnosti v grafih. Za odkrivanje skupnosti ne potrebuje dodatnih funkcij ali informacij, uporabi le vozlišča in njihove povezave. Oznake potujejo po omrežju v več iteracijah, dokler ni nobene spremembe oznake vozlišča. Oznaka postane dominantna v gosto povezanih skupinah vozlišč. Vozlišča z istimi oznakami tvorijo skupnosti.
Časovna zahtevnost je skoraj linearna.

15.1 Algoritem širjenja oznak

Algoritem širjenja oznak iterativno priredi vsakemu vozlišču oznako najštevičnejšo med sosedi vozlišča na naslednji način [48]:
1. Vsako vozlišče dobi svojo enolično oznako
2. Iterativno širjenje oznak dokler
   ni doseženo uporabniško definirano največje število iteracij ali
   če nobeno vozlišče omrežja ne spremeni oznake.
- Oznake se razširjajo po omrežju za vsa vozlišča omrežja v vsaki iteraciji.
- Vrstni red obiska posameznega vozlišča je naključen.
- Označba obiskanega vozlišča se spremeni na označbo, ki jo ima največ sosedov.

> Ob razširjanju oznak dobijo gosto povezane skupine vozlišč hitro eno oznako. Na koncu ostane le nekaj oznak, večina jih izgine. Vozlišča z isto oznako tvorijo skupnost.

Algoritem razširjanja oznak ima prednosti in pomanjkljivosti, kar opišejo že avtorji [48]:

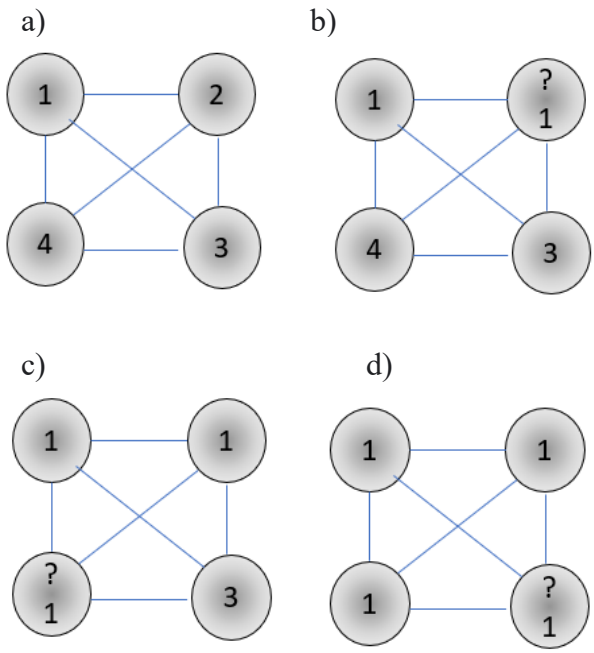
- Prednost je skoraj linearna časovna zahtevnost.

- Slabost algoritma je, da lahko za iste začetne vrednosti dobi različne strukture gruč. To je posledica različnega naključnega zaporedja obdelave vozlišč gruč in tudi naključne izbire sosedne oznake skupnosti vozlišča v primerih, ko sta dve oznaki sosedov enako številčni.

Območje različnih odkritih skupnosti v omrežju lahko zožimo, če oznake podelimo samo nekaterim vozliščem, ostala vozlišča pa pustimo brez oznak.

15.2 Primer

Slika 24 prikazuje primer štirih iteracij širjenja označb v grafu s štirimi vozlišči.



Slika 24: Štirih iteracije (a, b, c, d) širjenja označb v grafu s štirimi vozlišči.

16 Napoved povezav

V nadaljevanju opišemo dva algoritma za napovedovanje povezav v knjižnici Neo4j Graph Data Science. Algoritmi napovedi povezav določajo bližino parov vozlišč, ki jo lahko uporabimo za napoved povezav med vozlišči.

16.1 Algoritem Adamic Adar

Algoritem Adamic Adar [49] sta leta 2003 uvedla Lada Adamic and Eytan Adar za napovedi povezav v socialnih omrežjih. Za merilo povezav uporabita naslednjo enačbo:

$$AA(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log|N(u)|}$$

kjer je N(u) množica sosedov vozlišča u. Vrednost 0 označuje dve oddaljeni, nepodobni vozlišči, večje vrednosti pa označujejo podobna vozlišča.

16.2 Skupni sosedi

Zelo razširjeno merilo za oceno povezanosti dveh vozlišč je tudi število skupnih sosedov:

$$SS(x,y) = |N(x) \cap N(y)|$$

kjer je *N(x)* množica sosedov vozlišča *x* in *N(y)* množica sosedov vozlišča *y* . Kot pri prejšnjem algoritmu, večje vrednosti označujejo podobna vozlišča, vrednosti blizu 0 pa nepodobna vozlišča.

16.3 Prednostno pripenjanje

Algoritem sta predlagala Albert-László Barabási in Réka Albert [51] z generiranjem naključnih razširljivih omrežij (angl. scale-free networks).

$$PP(x,y) = |N(x)| * |N(y)|$$

kjer je *N(x)* množica sosedov vozlišča *x* in *N(y)* množica sosedov vozlišča *y*. Kot pri ostalih metodah za napoved povezav, večje vrednosti PP(x, y) nakazujejo na večjo podobnost vozlišč.

16.4 Dodelitev virov

Algoritem dodelitve virov so leta 2009 predstavili Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang [52] za napoved povezav v različnih omrežjih. Uporablja naslednjo enačbo:

$$DV(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{|N(u)|}$$

Vrednost *DV(x,y)* 0 pomeni, da vozlišči x in y nista podobni in višje vrednosti pomenijo, da sta vozlišči podobni.

16.5 Število edinstvenih sosedov

Bližino sosedov določa tudi število edinstvenih sosedov dveh vozlišč x in y B(x,y). Temelji na ideji, večje je število sosedov obeh vozlišč, bolj povezani sta vozlišči, večja je možnost novih povezav.
$CN(x,y) = |N(x) \cup N(y)|$
Vrednost *CN(x,y) 0* pomeni, da vozlišči *x* in *y* nista podobni in višje vrednosti pomenijo, da sta vozlišči podobni.

16.6 Ista gruča

Neo4j Graph Data Science nudi funkcijo za napoved povezav poimenovano ista gruča [53]. Dve vozlišči omrežja iz iste gruče imata večjo verjetnost povezave med njima, če je še ni. Vrednost 0 pomeni, da vozlišč nista v isti gruči, vrednost 1 pa imajo vozlišča, ki so v isti gruči.

17 Literatura

[1] Codd, E.F. (1970). A relational model of data for large shared data banks. *M.D. computing: computers in medical practice, 15 3*, 162-6.

[2] Query Optimizer Concepts, https://docs.oracle.com/en/database/oracle/oracle-database/21/tgsql/query-optimizer-concepts.html#GUID-06129ACE-36B2-4534-AE68-EDFCAEBB3B5D; v Oracle® Database SQL Tuning Guide, 21c, F31828-05, August 2021.

[3] Bayer/McCreight: Organization and Maintenance of Large Ordered Indices. Acta Informatica 1(3), 173–189, 1972.

[4] 5. Indexes and Index-Organized Tables, B-Tree indexes, https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/indexes-and-index-organized-tables.html#GUID-797E49E6-2DCE-4FD4-8E4A-6E761F1383D1; v: Oracle® Database, Database Concepts, 21c,F31733-04 August 2021.

[5] 9. Joins, https://docs.oracle.com/en/database/oracle/oracle-database/21/tgsql/joins.html#GUID-BD96F1B4-76D4-43DF-98B6-D07F46838C4A; v: Oracle® Database, SQL Tuning Guide, 21c, F31828-05, August 2021.

[6] 9.2.1 Nested Loops Joins, https://docs.oracle.com/en/database/oracle/oracle-database/21/tgsql/joins.html#GUID-A2DA6A1E-6180-4AB9-A777-586AF3953D53; v: Oracle® Database, SQL Tuning Guide, 21c, F31828-05, August 2021.

[7] Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, McGraw-Hill, 3rd ed., 2007.

[8] 9.2.2. Hash joins, https://docs.oracle.com/en/database/oracle/oracle-database/21/tgsql/joins.html#GUID-91E61BDC-E5F2-49FA-99AE-DD88A2FBB4FB; v Oracle® Database, SQL Tuning Guide, 21c, F31828-05, August 2021.

[9] 9.2.3. Sort Merge Joins, https://docs.oracle.com/database/121/TGSQL/tgsql_join.htm#GUID-5FCD34FE-ED04-4AB2-BC90-9752FED94F4F; v: **Oracle® Database,** SQL Tuning Guide, 21c, F31828-05, August 2021.

[10] Part II, Machine Learning Functions, https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmcon/oracle-machine-learning-sql-concepts.pdf (v: Oracle® Machine Learning for SQL Concepts,21c F31930-05, June 2021).

[11] Part III, Algorithms, https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmcon/oracle-machine-learning-sql-concepts.pdf; v: Oracle® Machine Learning for SQL Concepts,, 21c F31930-05, June 2021.

[12] Cluster Analysis: Basic Concepts and Algorithms (chapter 8), https://www-users.cse.umn.edu/~kumar001/dmbook/ch8.pdf; v: Tan P.-N., Steinbach M. in Kumar V. Introduction to Data Mining, Pearson Addison Wesley, 2006.

[13] Campos, M.M., Milenova, B.L., Clustering Large Databases with Numeric and Nominal Values Using Orthogonal Projections, Oracle Data Mining Technologies, Oracle Corporation, 1997.
https://www.researchgate.net/publication/228726561_Clustering_large_databases_with_numeric_and_nominal_values_using_orthogonal_projections#pf4

[14] R. Agrawal, R. Srikant, Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[15] Association Analysis: Basic Concepts and Algorithms (6 chapter) v: Tan P.-N., Steinbach M. in Kumar V. Introduction to Data Mining, Pearson Addison Wesley, 2006. https://www-users.cse.umn.edu/~kumar001/dmbook/ch6.pdf/

[16] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[17] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. **1**. University of California Press. pp. 281–297. MR 0214227. Zbl 0214.46201.

[18] 17 Expectation Maximization, v: Oracle® Machine Learning for SQL: Concepts, 21c, F31930-05, June 2021. https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmcon/expectation-maximization.html#GUID-F4D117F3-FA0C-4CA4-9034-67D12339AE90.

[19] 21 Support Vector Machines, v: Oracle® Machine Learning for SQL: Concepts, 21c, F31930-05, June 2021. https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmcon/support-vector-machine.html#GUID-FD5DF1FB-AAAA-4D4E-84A2-8F645F87C344.

[20] Joyce, James (2003), "Bayes' Theorem", in Zalta, Edward N. (ed.), The Stanford Encyclopedia of Philosophy (Spring 2019 ed.), Metaphysics Research Lab, Stanford University.

[21] 24. Naive Bayes, v: Oracle® Machine Learning for SQL: Concepts, 21c, F31930-05, June 2021. https://docs.oracle.com/en/database/oracle/machine-learning/oml4sql/21/dmcon/naive-bayes.html#GUID-BB77D68D-3E07-4522-ACB6-FD6723BDA92A.

[22] 5 Neural Network, v: Oracle® Machine Learning for SQL, Concepts, 21c, F31930-05, June 2021. https://docs.oracle.com/en/database/oracle/machine-

learning/oml4sql/21/dmcon/neural-network.html#GUID-C45971D9-A874-4546-A0EC-1FF25B229E2B .

[23] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[24] Kalita, H., Krishnaprasad, A., Choudhary, N. *et al.* Artificial Neuron using Vertical MoS_2/Graphene Threshold Switching Memristors. *Sci Rep* **9,** 53 (2019). https://doi.org/10.1038/s41598-018-35828-z.

[25] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". *Neural Networks*. **61**: 85–117. https://arxiv.org/pdf/1404.7828.pdf

[26] Neo4j Graph Data Science- Algorithm; v: The Neo4j Graph Data Science Library Manual v1.7, 2021 Neo4j Inc.,
https://neo4j.com/docs/graph-data-science/current/algorithms/ .

[27] Neo4J graph data platform: The fastest path to the graph. https://neo4j.com/.

[28] Node Similarity, v: The Neo4j Graph Data Science Library Manual v1.7, 2021 Neo4j Inc., https://neo4j.com/docs/graph-data-science/current/algorithms/node-similarity/

[29] Jaccard, P. (1912). The Distribution of the Flora of the Alpine Zone. New Phytologist, 11, 37-50.
http://dx.doi.org/10.1111/j.1469-8137.1912.tb05611.x

[30] Dong, W., Charikar, M., & Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. WWW '11: Proceedings of the 20th international conference on World wide web, 2011, 577–586.

[31] Brin, S.; Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems. 30 (1–7): 107–117. ISSN 0169-7552, https://doi.org/10.1016/S0169-7552(98)00110-X.

[32] Klicpera, J., Bojchevski, A., & Günnemann, S. (2019). Predict then Propagate: Graph Neural Networks meet Personalized PageRank. *ICLR*. https://arxiv.org/abs/1810.05997

[33] Zaki, Mohammed J.; Meira, Jr., Wagner (2014). Data Mining and Analysis: Fundamental Concepts and Algorithms. Cambridge University Press. ISBN 9780521766333.

[34] Freeman, Linton (1977). A set of measures of centrality based on betweenness. Sociometry. **40**(1): 35–41. doi:10.2307/3033543. JSTOR 3033543.

[35] Johnson, Donald B. (1977), Efficient algorithms for shortest paths in sparse networks, Journal of the ACM, 24 (1): 1–13, doi:10.1145/321992.321993.

[36] Bellman, Richard (1958). On a routing problem. Quarterly of Applied Mathematics. 16: 87–90. doi:10.1090/qam/102435. MR 0102435.

[37] Ford, Lester R. Jr. (August 14, 1956). Network Flow Theory. Paper P-923. Santa Monica, California: RAND Corporation.

[38] Moore, Edward F. The shortest path through a maze. Proc. Internat. Sympos. Switching Theory 1957, Part II. Cambridge, Massachusetts: Harvard Univ. Press. pp. 285–292. 1959. MR 0114710.

[39] Brandes, Ulrik. A faster algorithm for betweenness centrality (PDF). Journal of Mathematical Sociology. **25** (2): 2001. 163–177. doi:10.1080/0022250x.2001.9990249.

[40] Brandes, U., & Pich, C. Centrality Estimation in Large Networks. Int. J. Bifurc. Chaos, 17, 2007 2303-2318. https://www.uni-konstanz.de/mmsp/pubsys/publishedFiles/BrPi07.pdf

[41] Freeman, Linton C. Centrality in social networks conceptual clarification. Social networks 1.3. 1979: 215–239.

[42] Bavelas, Alex (1950). Communication Patterns in Task-Oriented Groups. The Journal of the Acoustical Society of America. **22** (6): 725–730. doi:10.1121/1.1906679.

[43] Krebs, V. E., Mapping Networks of Terrorist Cells. CONNECTIONS 24(3): 43-52 2002 INSNA.

[44] Borgatti, S. Centrality and network flow. Soc. Networks, 27, 2005, 55-71.

[45] Blondel, V.D., Guillaume, J., Lambiotte, R., & Lefebvre, E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008, 10008.

[46] Newman, M. E. J. Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America. **103**(23): 8577–8696. arXiv:physics/0602124.

[47] Blondel, Vincent D; Guillaume, Jean-Loup; Lambiotte, Renaud; Lefebvre, Etienne. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment. 2008 (10): P10008. arXiv:0803.0476.

[48] Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. Physical review E, 76(3), 036106.

[49] E. W. Dijksta, A Note on Two Problems in Connexion with Graphs, Numerische Mathematlk l, 269 – 271, l 959.

[50] Adamic, L.A., & Adar, E., Friends and neighbors on the Web. Soc. Networks, 25, 2003. 211-230.

[51] Barabási, Albert-László; Albert, Réka (October 1999). "Emergence of scaling in random networks" (PDF). Science. 286(5439): 509–512. arXiv:cond-mat/9910332.

[52] Zhou, T., Lü, L., & Zhang, Y. C. (2009). Predicting missing links via local information. The European Physical Journal B, 71(4), 623-630.

[53] Same Community. The Neo4j Graph Data Science Library Manual v1.7, 2021 Neo4j, Inc., https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/same-community/.