SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 911

KLASIFIKACIJA KODIRAJUĆIH REGIJA U GENOMU

Maja Štimac

Zagreb, Lipanj 2009.

Hvala mentoru prof.dr.sc. D. Seršiću na uloženom trudu i pomoći oko izrade ovog završnog rada, Dr.sc. Mile Šikiću na konstruktivnim savjetima te svim ostalim asistentima i profesorima.

Sadržaj

1. UVOD

Razvojem tehnologije područje proučavanja gena, njihove strukture te uloge u pojedinim stanicama svakodnevno dolazi do novih otkrića. Broj genoma u javno dostupnim baza stalno je u porastu. Veliki interes za ovo područje dolazi i od činjenice da je DNK osnovna molekula na kojoj se temelji nasljeđivanje te je odgovorna za prenošenje nasljednog materijala i osobina. Mijenjanjem njezinog sadržaja nastaju mutacije koje mogu uzrokovati različite bolesti. Poznavanje i razumijevanje uloga pojedinih dijelova DNK ključni su u borbi protiv tih bolesti.

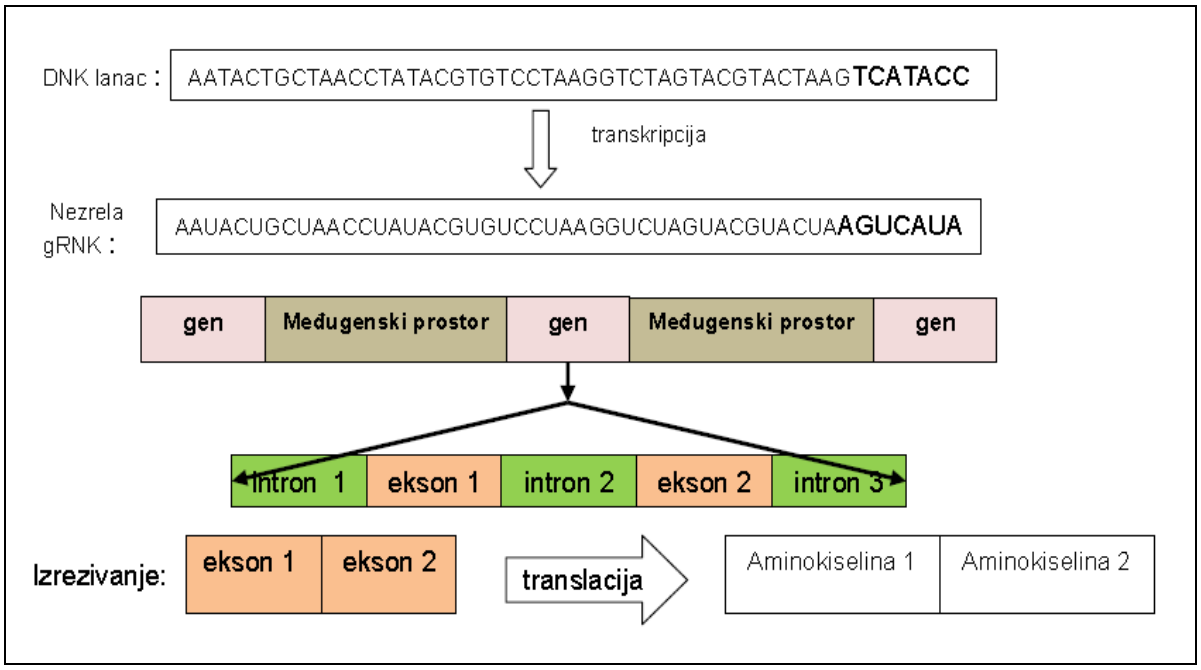
Za dokazivanje mjesta izrezivanja introna odnosno lokacija eksona, učestalost transkripcije te aktivnost pojedinog gena u određenim stanicama, uvijek će biti potrebno provođenje molekularnih eksperimenata, no danas se pri obradi podataka sve više koriste računalne metode. One mogu ponuditi učinkovitije razvrstavanja velikih količina podataka, lakše uočavanje dijelova DNK molekule koji se možda ponavljaju te indicirati mogući položaj eksona unutar gena. Na taj način se želi ubrzati postupak traženja gena i točnih mjesta izrezivanja te smanjiti opseg eksperimenata i njihovih troškova. Područje znanosti u kojem se objedinjuju znanja iz biologije, računalne znanosti te matematike u jednu disciplinu naziva se Bioinformatika.

2. BIOLOŠKE OSNOVE

Deoksiribonukleinska kiselina ili DNK molekula sastoji se od dva spiralno povezana komplementarna DNK lanca. Lanac je sastavljen od četiri nukleotida ili baza: adenina (A), timina (T), citozina(C) i guanina(G). Unutar DNK molekule lanci se povezuju tako da se adenin veže s timinom, a guanin s citozinom. Svaki DNK lanac sadrži gene i međugenski prostor (područja koja reguliraju gene i područja s još nepoznatim funkcijama). Gen se sastoji od eksona i introna. Eksoni, kodirajuća područja, zaslužni su za stvaranje proteina dok introni, nekodirajuća područja, ne sudjeluju u tom procesu.

Protein je biomolekula koju tvori niz aminokiselina. Proteini su odgovorni za odvijanje i/ili ubrzavanje gotovo svih bioloških funkcija. Svaki protein sastoji se od jedinstvenog niza aminokiselina. Setovi od tri nukleotida nazivaju se kodoni. Svaka kombinacija od tri nukleotida predstavlja neku aminokiselinu. Broj postojećih aminokiselina je 20 tako da nekoliko kodona može kodirati jednu aminokiselinu.

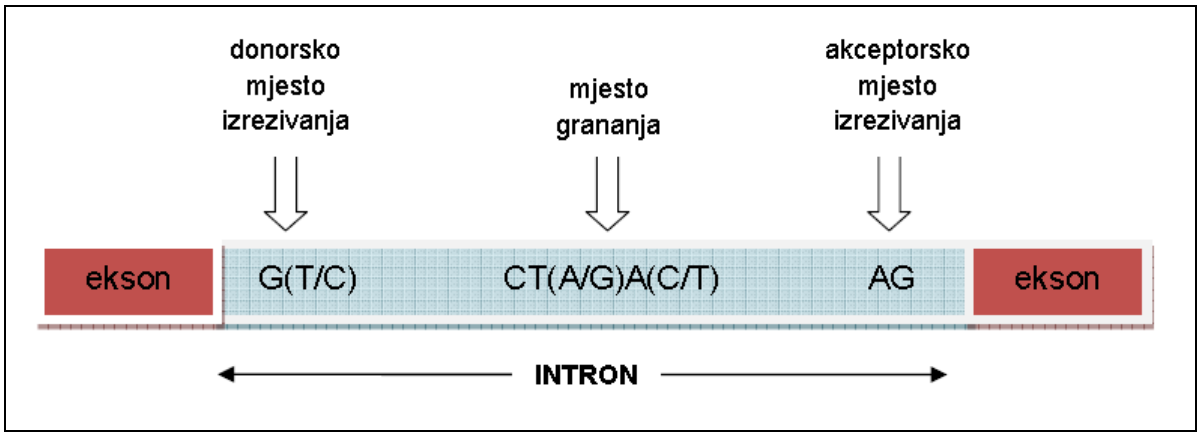
Proces sinteze proteina može se podijeliti na tri glavna koraka: Transkripcija, Izrezivanje (*eng. Splicing)* i Translacija.



Slika 1: Sinteza proteina

Transkripcija je proces stvaranja nezrele RNA molekule (mRNA) koja nastaje preslikavanjem DNK lanca uz zamjenu timina s uracilom. Transkripcija se odvija u jezgri, a dobivena mRNA ulazi u proces izrezivanja. Regije koje se uklanjaju u procesu izrezivanja RNA su introni, čime se eksoni spajaju jedan na drugi. Tako dobivena zrela glasnička RNA prenosi se iz jezgre u citosol, gdje ribosomi započinju prevođenje (translaciju) RNA u proteine.

Usprkos poznavanju potpunog genoma određenih organizama, pozicije kodirajućih eksona i nekodirajućih introna još uvijek su nepoznate. Poznato je da se na početku svakog introna u 99% slučajeva nalazi kombinacija nukleotida G-T, a u 1% slučajeva kombinacija G-C, dok se na početku svakog eksona nalazi kombinacija A-G. Početak introna naziva se još i donorsko mjesto izrezivanja (*eng. donor splice site*), a kraj svakog introna zove se akceptorsko mjesto izrezivanja (*eng. acceptor splice site*). Mjesto grananja (eng. *Branch point*) je oblika CT(A/G)A(C/T).



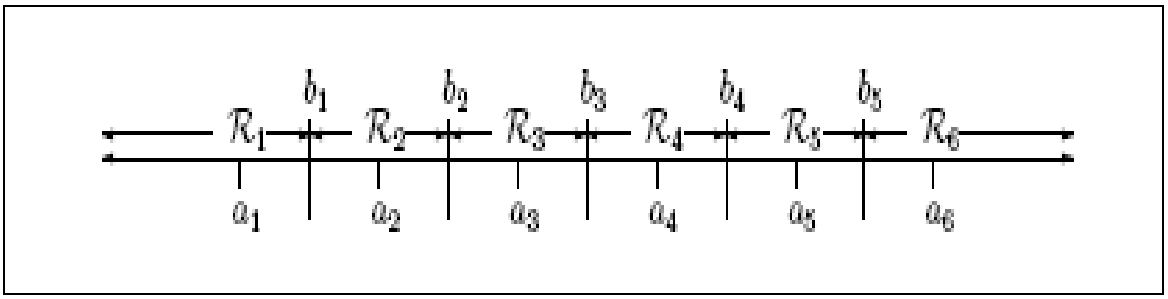
Slika 2: Donorsko i akceptorsko mjesto izrezivanja introna

3. VEKTORSKA KVANTIZACIJA

3.1. LLOYD-MAX KVANTIZATOR

Način rada ovog kvantizatora najjednostavnije je opisati na kvantizaciji skalara, a kvantizacija n-dimenzionalnih vektora provodi se na analogan način.

Kvantizator skalara dijeli skup realnih brojeva u M podskupova R_1 do R_M. Svako od tih kvantizacijskih područja R_j predstavlja jedna kvantizacijska razina a_j, koja je i sama realni broj. Realni broj x koji pripada podskupu R_j tada se kvantizira na vrijednost a_j. Osnovni problem je kako odabrati kvantizacijska područja i kvantizacijske razine da bi srednja kvadratna pogreška kvantizacije bila minimalna. Na slici 3. prikazana su kvantizacijska područja i razine za M = 6, pri čemu b_1, ..., b_{M-1} predstavljaju granice intervala.



Slika 3. Kvantizacijska područja i razine za M = 6

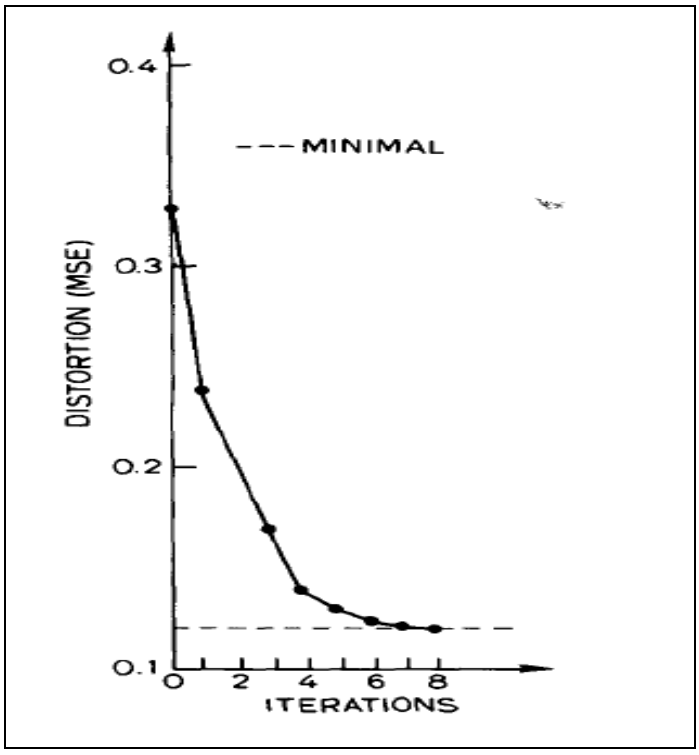
Postavljaju se dva pitanja: kako za određeni skup kvantizacijskih razina $\{a_j\}$ odrediti kvantizacijska područja $\{R_j\}$ te kako za određeni skup kvantizacijskih područja $\{R_j\}$ odrediti kvantizacijske razine $\{a_j\}$. Da bi se minimizirala pogreška kvantizacije, granica b_j između intervala R_j i R_{j+1} mora ležati na pola puta između a_j i a_{j+1}. Ovo ne ovisi o vjerojatnosti pojavljivanja pojedine vrijednosti x. Neka funkcija $Q(x)$ preslikava sve vrijednosti $x \in R_j$ u a_j i neka je $f_x(x)$ funkcija gustoće vjerojatnosti od x. Tada srednja kvadratna pogreška iznosi:

$$E[(x-Q(x))]^2] = \int_{-\infty}^{\infty} f_x(x) * (x-Q(x))^2 \, dx = \sum_{j=1}^{M} \int_{R_j} f_x(x) * (x-a_j)^2 \, dx \qquad (1)$$

Za pojedini interval R_j dobije se da je srednja kvadratna pogreška minimalna za

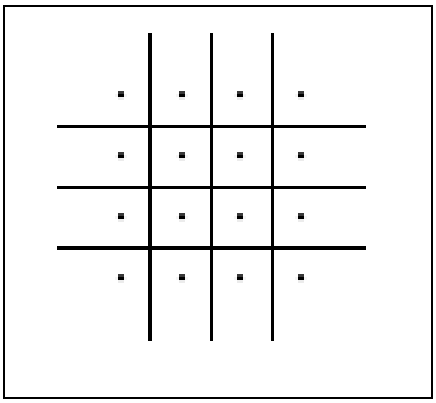
$$a_j = \bar{x}_j \qquad (2)$$

Lloyd-Max algoritam optimizira prvo granice intervala $\{b_j\}$ za određene $\{a_j\}$, a potom određuje nove razine $\{a_j\}$ za dobivene granice $\{b_j\}$. Ovaj postupak se ponavlja sve dok se srednja kvadratna pogreška ne smanji na neku određenu vrijednost. Nakon određenog broja ponavljanja, pogreška se dalje neznatno smanjuje. Ovisnost srednje kvadratne pogreške (distorzije) o broju iteracija prikazana je na slici 4. Crtkanom linijom prikazana je minimalna kvadratna pogreška.



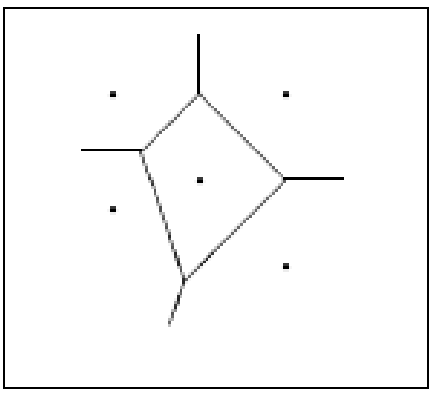
Slika 4. Ovisnost srednje kvadratne pogreške o broju ponavljanja

Kvantizacija *n*-dimenzionalnog vektora svodi se na kvantiziranje *n* varijabli. Moguće je svaku dimeziju kvantizirati posebno, kao skalar, čime se dobije pravokutna mreža kvantizacijskih područja uz srednju kvadratnu pogrešku svake dimenzije jednaku onoj skalara (na slici 5 prikazan je slučaj n = 2).



Slika 5. Dvodimenzionalna kvantizacija, svaka dimenzija posebno

Optimalnu (istovremenu) kvantizaciju više varijabli najjednostavnije je opisati na primjeru *n* = 2. Odabere se *M* dvodimenzionalnih kvantizacijskih točaka {(a_j, a_j')} te se za neku točku (x, x') odabire takva (a_j, a_j') koja joj je najbliža po euklidskoj udaljenosti (odnosno za koju je $d = \sqrt{(x - a_j)^2 + (u' - x_j')^2}$ najmanji). Područja R_j potom se odabiru kao skup točaka (x, x') koje su najbliže upravo tom određenom (a_j, a_j'). Granica između svaka dva područja je okomica na spoj njihovih kvantizacijskih točaka (slika 6). Lloyd-Max algoritam moguće je na ekvivalentan način jednostavno provesti kako za dvodimenzionalni, tako i za bilo koji *n*-dimenzionalni slučaj.



Slika 6. Dvodimenzionalna kvantizacija, obje dimenzije zajedno

3.2. LINDE-BUZO-GRAY CLUSTERING ALGORITAM

Generalizacijom optimalnog skalarnog Lloyd-Max kvantizatora nastao je Linde-Buzo-Gray algoritam kojim ćemo se služiti u ovom radu.

3.2.1. OPIS RADA ALGORITMA

Ako je $X_i=[x_{i1}\ x_{i2}\ x_{i3}\ ...\ x_{ip}]$ i-ti ulazni vektor, a $W_i=[w_{i1}\ w_{i2}\ w_{i3}\ ...\ w_{ip}]$ njegova pripadna težinska funkcija, tada se WED udaljenost ovog vektora do nekog drugog (taj drugi može biti centroid) $Y=[y_1\ y_2\ y_3\ ...\ y_p]$ definira kao:

$$d_2(X_i,Y)=\sum_{j=1}^{p}(x_{ij}-y_j)^2 * w_{ij} \qquad (3)$$

Ovaj d_2 naglašava da se radi o kvadratnoj mjeri udaljenosti. Ako imamo skup vektora X_1 do X_N, tada je za taj skup vektora moguće naći centroid koji minimizira sumu WED udaljenosti svih ulaznih vektora prema centroidu:

$$\min(Y)=\sum_{i=1}^{N} d_2\,(X_i,Y) \qquad (4)$$

Y se nalazi kao težinska srednja vrijednost od X_i

$$y_j=\frac{\sum_{i=1}^{N} w_{ij} * x_{ij}}{\sum_{i=1}^{N} w_{ij}} \quad za\ j=1..P \qquad (5)$$

Ova modifikacija koristi se u izračunavanju distorzije i centroida. S obzirom da je mjera distorzije kompatibilna s postupkom izračunavanja centroida, ova modifikacija ne utječe na konvergenciju postupka.

Originalna euklid rutina izračunava korijen iz kvadrata razlike komponenata što odgovara pravoj udaljenosti, dok ova modifikacija računa kvadrat euklidske udaljenosti i srednje vrijednosti. Usrednjavanje korijenske udaljenosti nije odgovaralo korištenoj metodi računanja centroida (obično težište) jer se kao mjera distorzije morala koristiti srednja kvadratna pogreška. Njezinim minimiziranjem određivao se novi centroid. Korištenjem korijenske udaljenosti događalo se da se nakon izračunavanja novih centroida prosječna udaljenost povećala, ali prosječna kvadratna udaljenost smanjila. To je uzrokovalo da se postupak određivanja broja grupa (*eng. clustering*) prekine jer umjesto da se greška smanjuje, ona raste.

3.2.2. METODA FORMIRANJA NOVOG CENTROIDA

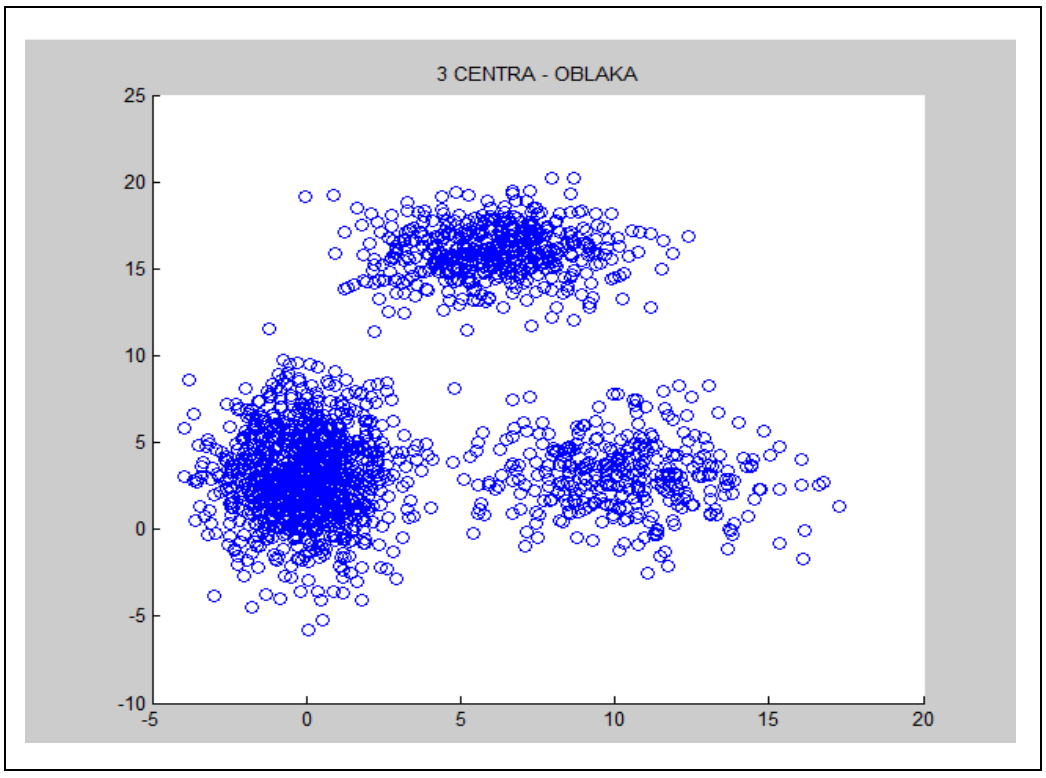
Funkcija za vektorsku kvantizaciju učitava ulazni niz podataka. Na temelju formule za težište pronalazi centroid ulaznog niza. Prvobitna ćelija (centroid) se raspada na dvije nove odabiranjem rogova prvobitne ćelije koji postaju novi centroidi. Postupak formiranja novog centroida:

- ➔ Ako su u ćeliji manje od dva elementa tada nije moguće formirati novi centroid
- ➔ Ako su u ćeliji točno dva elementa tada svaki postaje jedan centroid
- ➔ Ako ima 3 ili više elementa, tada:
    1. za sve elemente te ćelije pronalazi se udaljenost do starog centroida
    2. kao jedan od novih centroida odberi onaj element koji najudaljeniji od starog centroida
    3. ponovno za sve elemente te ćelije pronalazi se udaljenost do prvog novog centroida (kojeg smo pronašli u prethodnom koraku)
    4. kao drugi novi centroid odabire se onaj element koji je najudaljeniji od prvog novog centroida

3.2.3. POSTUPAK TRAŽENJA KOLJENA DISTORZIJE

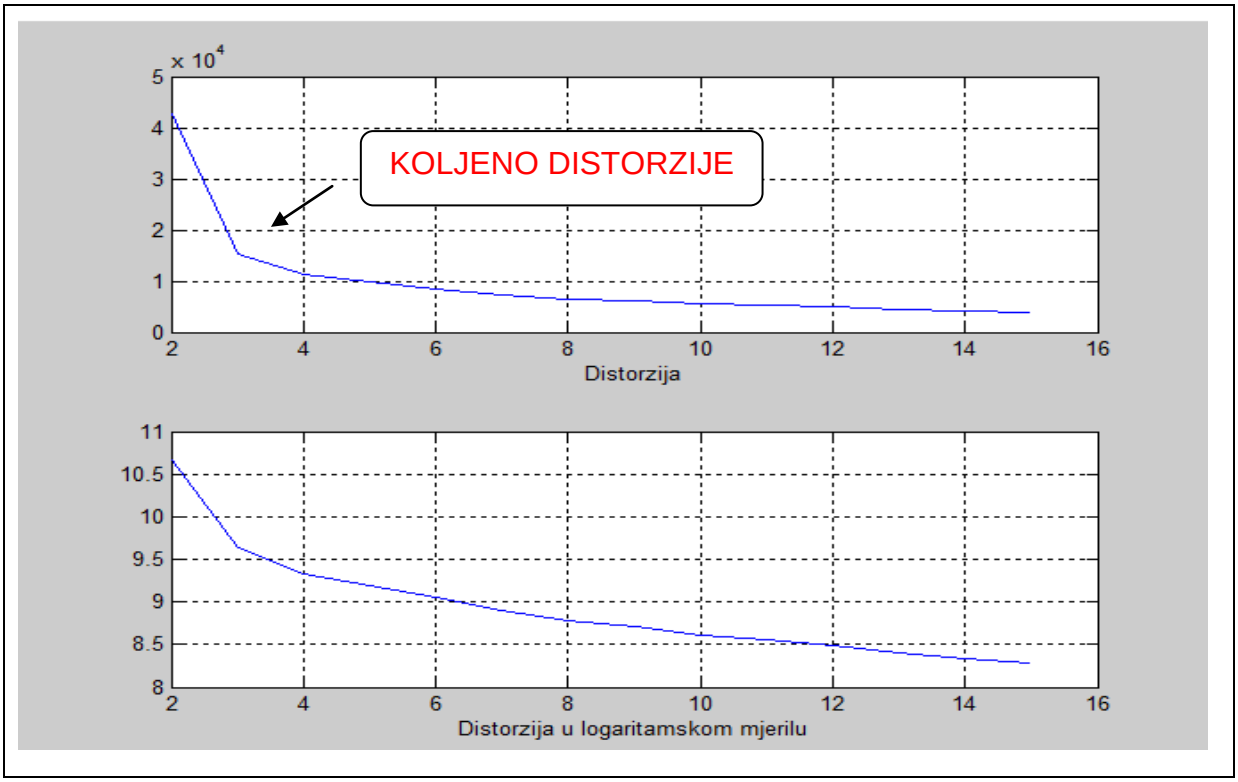
Postupak dobivanja koljena distorzije korištenjem Linde-Buzo-Gray algoritma radi jednostavnosti objašnjen je prvo u dvodimenzionalnom prostoru, a nakon toga će se povezati sa stvarnim podacima u višedimenzionalnom prostoru. Vrijednosti vektora X_1 i X_2 dobiju se slučajnim odabirom zadanog broja točaka, određenog raspršenja oko neke vrijednosti. Sljedeći kod demonstrira način dobivanja tih vektora i prikaz u 2D prostoru.

```
% >> 3 CENTRA <<
% Centri:[0,3], [6,16], [10, +3]
% Rasprsenja:[1.5,2.5], [2, 1.5], [2.5, 2]
x1 = [1.5*randn(1000,1)+0; 2*randn(500,1)+6;    2.5*randn(350,1)+10];
x2 = [2.5*randn(1000,1)+3; 1.5*randn(500,1)+16; 2*randn(350,1)+3];
% Graficki prikaz
figure, scatter(x1, x2);  % prikaz 2D podataka
```



Slika 7: Prikaz vektora u 2D prostoru

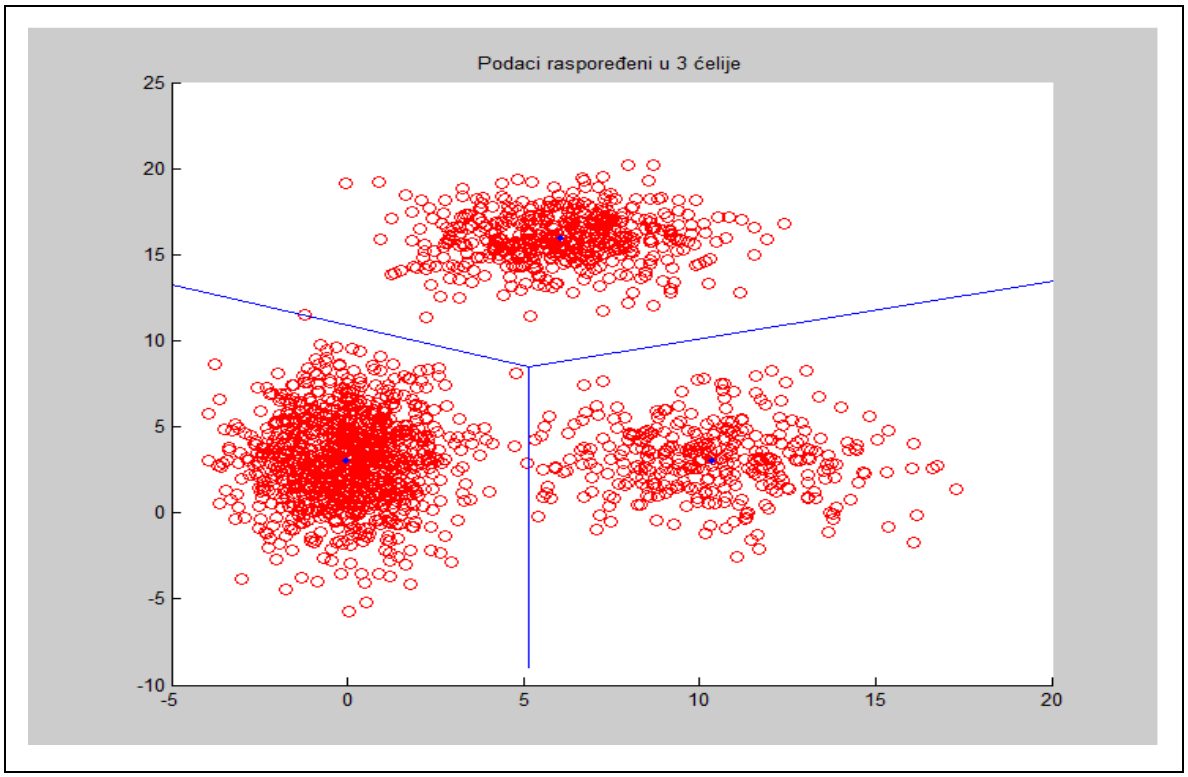
Promatranjem slike uočavaju se tri nakupine točaka, slikovito nazvane oblacima. Korištenjem navedenog algoritma ta tri oblaka se vrlo lako mogu podijeliti u tri ćelije uz prethodno zadavanje broja elemenata kodne tablice (u ovom slučaju Muk=3). S obzirom da broj oblaka neće uvijek biti poznat pokazalo se da se on može vrlo pouzdano odrediti pomoću koljena distorzije. Mijenjanjem broja kvanata (Muk={2,3,4,5,6,7,8,9,10,11…}) mijenja se i broj članova vektora distorzije. Za tri centra, vektor distorzije će imati tri člana. Za svaki kvant izračunava se i zapisuje sumarna distrozija (sum(distorzije)) na temelju koje se crta krivulja distorzije. Ako je broj oblaka tri očekujemo i koljeno distorzije za Muk=3. Treba uočiti da za broj kvanata veći od broja centara distorzija se smanjuje po poznatom zakonu, dok to nije slučaj za manji broj kvanata.



Slika 8: Krivulja distorzije

Na slici je prikazana krivulja distorzije u dva mjerila. Iako bi krivulja u logaritamskom mjerilu trebala biti linearnija u praksi se pokazalo da je nekad lakše uočiti koljeno distorzije s aritmetičkog, a nekad s logaritamskog mjerila.

Iako je prostim oko lako vidljivo koljeno distorzije, Matlab sprema nagibe distorzije u matricu, radi omjere susjednih nagiba te ih oduzima i uspoređuje. Najveća razlika predstavlja koljeno distorzije. Nakon što je pronađeno koljeno, postaje poznat i broj oblaka odnosno centroida na temelju kojeg se ulazni skup podataka lako raspodjeljuje u ćelije.



Slika 9: Podjela oblaka u ćelije

4. REZULTATI ISTRAŽIVANJA

Ovaj rad temelji se na obradi podataka organizma Caenorhabditis elegans preuzetih sa stranice njemačkog instituta za biomedicinu („*Friedrich Miescher Laboratory*“). Iako postoje dva mjesta izrezivanja introna (donor i akceptor), ovdje će se obrađivati samo podaci vezani uz akceptorsko mjesto izrezivanja. Preuzeti podaci („*C_elegans_acc_all_examples.fasta*“) imaju ekstenziju („.fasta“) što je uobičajeni format organizacije podataka u biomedicini. Programski jezik koji se upotrebljava pri obradi podataka je Matlab. Statistička metoda koja se koristi su Slučajne šume (*eng. Random Forest*).

Ulazni skup podataka preuzet sa stranice njemačkog instituta veličine je 808 MB u fasta obliku. Podaci koje primaju RF su u .arff formatu (*eng. Attribute-Relation File Format*). Sve datoteke s tom ekstenzijom imaju točno definirano zaglavlje koje sadrži ime datoteke, atribute te redak koji upućuje na početak podataka. Nakon obrade podataka i pripremanja za rad s RF veličina podataka naraste na 1.4 GB. Zbog nedostatka memorije Matlaba i RF-a, podaci su morali biti smanjeni. Ovdje će se koristi ulazni set podataka koji sadrži sve točne i decimirane netočne, dok je duljina vektora smanjena s 398 na 142 nukleotida.

4.1. POSTUPAK PRONALAŽENJA KOLJENA DISTORZIJE

Prvi korak u pronalaženju koljena distorzije je obrada preuzetih podataka. Svaki vektor redak predstavlja jedno moguće mjesto izrezivanja odnosno mjesto u kojem se pojavila kombinacija nukleotida „AG“ (akceptorsko mjesto izrezivanja). Na kraju retka obično se stavljaju i brojni drugi parametri koju su vezani uz samo mjesto pronalaska tih nukleotida. Programom „Samo_tocni_acc.m“ izdvajaju se u novu datoteku („*C_elegans_samo_tocni_acc.txt*“) samo točna mjesta izrezivanja. Duljina vektora koja se pritom upisuje je 398 nukleotida, a na kraju se ne nalazi oznaka „T“ ili „F“ koja označava točnost mjesta izrezivanja s obzirom da su sva mjesta točna.

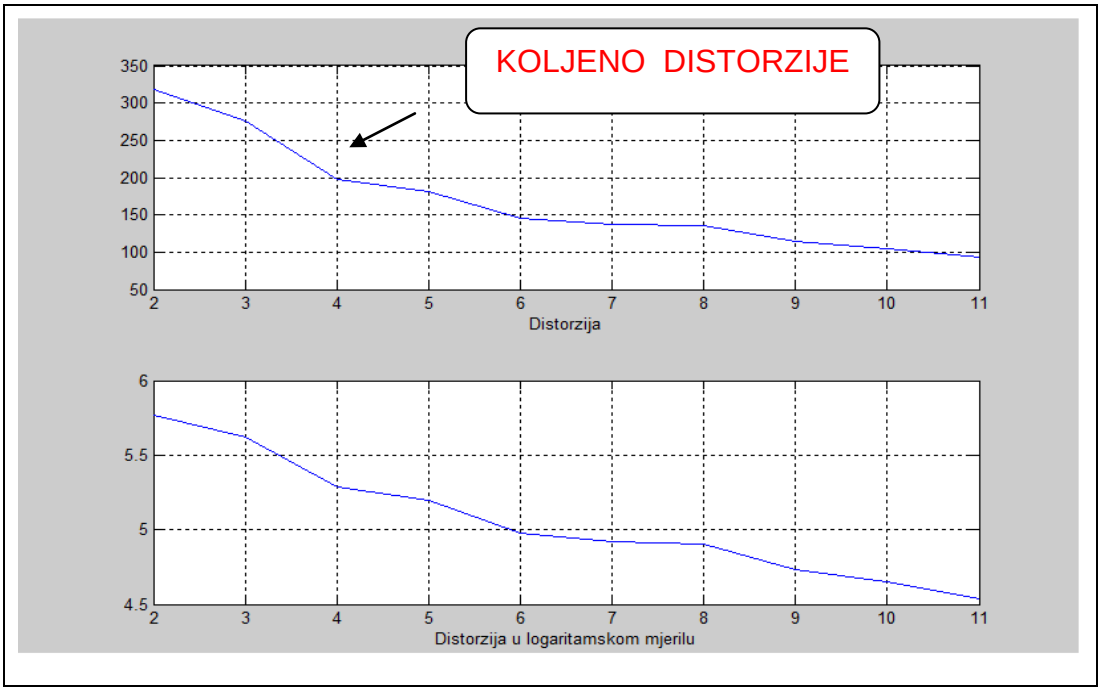
U postupku traženja koljena distorzije koristi se Linde-Buzo-Gray algoritam koji se poziva iz programa „Klaster.m“. Opis programa „Klaster.m“:

- Podaci iz datoteke sa samo točnim mjestima izrezivanja upisuju se u matricu_podataka
- Prije izvođenja algoritma za vektorsku kvantizaciju potrebno je učitane podatke prebaciti u višedimenzionalni prostor. S obzirom da postoje 4 različita slova koja moraju biti jednako razmaknuta u prostoru, koordinate koje dodjeljujemo svakom od njih su:

  A=[1 0 0], C=[0 1 0], G=[0 0 1], T=[-1/3 -1/3 -1/3]

- Kod C. Elegansa otkriven je slijed od 5 nukleotida koji se uvijek javlja ispred akceptorskog mjesta izrezivanja. Ta činjenica je iskorištena u ovom radu tako da nova matrica (X=[x y z];) podataka sadrži samo tih 7 slova (5+AG) svakog retka koji su prebačeni u višedimenzionalni prostor. Dimenzije nove matrice su X=64838x21 što znači da je u ulaznom skupu podataka *(„C_elegans_samo_tocni_acc.txt“)* pronađeno 64838 točnih mjesta izrezivanja, a broj dimenzija koji se koristiti je 21. S obzirom da je nemoguće slikom predočiti toliki broj dimenzija, a time i prostim okom vidjeti koliko ima nakupina odnosno oblaka jedini način kojim se može ustvrditi koliko postoji načina točnih mjesta izrezivanja je preko koljena distorzije.

- Pozivamo funkciju za vektorsku kvantizaciju. Na temelju ulaznog skupa podataka dobivamo sljedeću sliku.



Slika 10: Krivulja distorzije temeljena na samo točnim mjestima izrezivanja

- Koristeći kao mjeru udaljenosti/izobličenja euklidsku udaljenost, algoritam je ulazne vektore podijelio tako da pripadaju najbližem centru. Sa slike je vidljivo da je koljeno distorzije na Muk=4, stoga možemo zaključiti da postoji četiri različita načina izrezivanja točnih mjesta.

- Nakon pronalaska koljena distorzije potrebno je još jednom pozvati funkciju za vektorsku kvantizaciju, sada za poznati broja elemenata kodne tablice (kvanata) kako bi se izračunale izlazne varijable koje će se koristiti u sljedećim koracima.

4.2. RASPODJELA PODATAKA U SKUPINE

Iz koljena distorzije možemo zaključiti da se točna mjesta izrezivanja grupiraju u 4 oblaka (Muk=4). Sljedeći korak je razvrstati točna mjesta u skupine ovisno o tome kojem oblaku pripadaju. Opis programa Razvrstavanje.m:

Značenje pojedinih izlaznih varijabli dobivenih iz Linde-Buzo-Gray algoritma

- Varijabla BrojVektora nam kaže koliko vektora pripada kojem centru
- Varijabla pripadnost označava koji redak ulaznog skupa podataka pripada kojem centru
- Varijabla in predstavlja vektor stupac s indeksima matrice_podataka grupirane po skupinama (*eng.cluster*). Prva skupina sadrži sve indekse matrice_podataka koji pripadaju toj skupini idući prema rastućim predznacima nakon toga ide druga skupina koja sadrži sve indekse matrice_podataka koji pripadaju toj skupini itd.

Program na temelju prethodno navedenih varijabli razvrstava ulazne vektore koji se nalaze u matrici_podataka u četiri datoteke.

4.3. POSTUPAK DECIMIRANJA ULAZNOG SKUPA PODATAKA

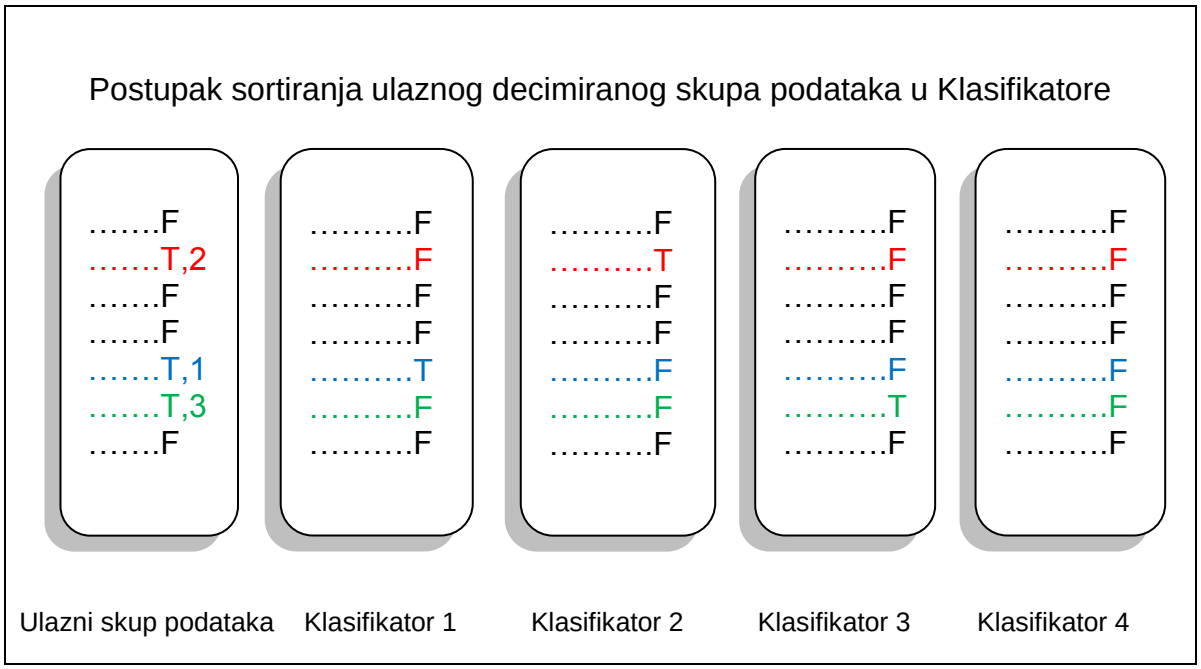
Zbog nedostatka memorije Matlaba i Slučajnih šuma pri radu s ovako velikom količinom podataka, ulazni skup po koji je preuzet sa stranice Njemačkog instituta za biomedicinu („*C_elegans_acc_all_examples.fasta*“) je smanjen tako da su iz njega uzeta sva točna i svako deseto netočno mjesto izrezivanja. Također duljina vektora jednog retka je smanjena na 142 slova metodom 80:60. Tom metodom uzimamo 80 slova prije AG i 60 slova nakon (uključujući i ta dva slova).

Točna mjesta izrezivanja se ispituju kako bi se ustvrdilo kojoj skupini pripadaju. Kada bi se uspoređivao cijeli redak ulazne datoteke sa svakim retkom iz sve četiri skupine to bi jako dugo. Stoga se uspoređivanje vršilo na temelju samo sedam slova (5+AG) odnosno pet slova ispred akceptorskog mjesta izrezivanja za koja znamo da su bitna u samom procesu izrezivanja. Na taj način se puno lakše moglo odrediti kojoj skupini pripada određeni redak ulazne datoteke.

Na kraju netočnih mjesta izrezivanja nalazi se oznaka „F“, a na kraju točnih „T“ i oznaka kojoj od prethodno definirane četiri skupine taj redak pripada. Program kojim se obavlja postupak decimiranja ulaznog skupa podataka naziva se Tocni_netocni(decimirani).m, a datoteka koju dobivamo *(„C_elegans_tocni+decimirani_netocni_acc.txt”)*.

4.4. POSTUPAK SORTIRANJA KLASIFIKATORE

Postupak dobivanja klasifikatora započinje čitanjem datoteke („*C_elegans_tocni+decimirani_netocni_acc.txt”*). Ukoliko je mjesto izrezivanja netočno ono se samo prepisuje, dok se kod točnih ispituje zadnje slovo u retku koje označava kojem klasifikatoru pripada to mjesto izrezivanja. Ako je mjesto izrezivanja pripadalo skupini 1 tada se taj red upisuje u klasifikator 1 s oznakom „T“ na kraju, te u sve ostale klasifikatore s oznakom „F“ na kraju. Odnosno taj se redak upisuje kao točno mjesto izrezivanja samo u onaj klasifikator kojem pripada, dok se u sve ostale upisuje kao netočno mjesto izrezivanja. Sljedeća slika ilustrira postupak sortiranja ulaznog decimiranog skupa podataka u klasifikatore.



Slika 11: Postupak sortiranja ulaznog decimiranog skupa podataka u Klasifikatore

4.5. PRIPREMANJE PODATAKA ZA SLUČAJNE ŠUME

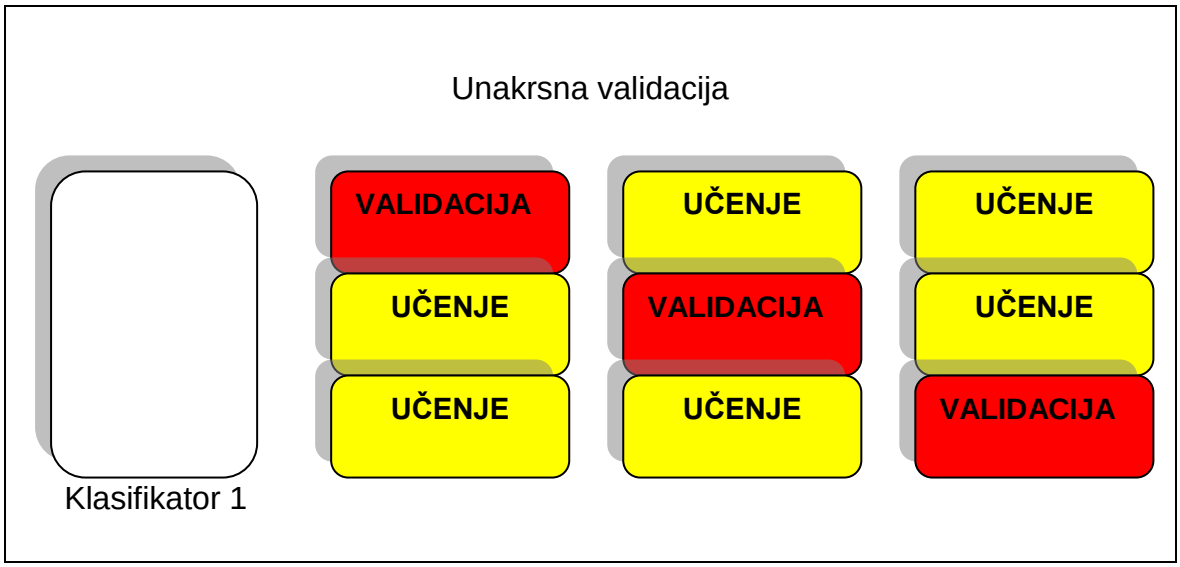
Slučajne šume (*eng. „Random Forest“)* su statistička metoda uzorkovanja s ponavljanjem. Iz ulaznog skupa se nasumično bira određeni broj podataka kreirajući tako jedno stablo. Istim principom dodjeljuju se drugim stablima njihovi podaci. Korisnik sam određuje broj stabala (porastom broja stabala smanjuje se vjerojatnost pogreške). RF radi na dva načina. U prvom koristi isti skup podataka za treniranje i za validaciju, a u drugom korisnik može sam odrediti koji skup želi koristiti za jedno, a koji za drugo. U fazi treniranja stabla se uče na poznatim podacima, dok u fazi validacije (out-of-bag podaci - koriste se za procjenu pogreške stabala) moraju donijeti ispravnu odluku na temelju nepoznatih podataka. Svako stablo glasa za ili protiv neke ulazne pretpostavke te se na temelju većinskih glasova (broj glasova je veći od zadanog praga) donosi odluka. U ovom radu RF su radile s unaprijed određenim podacima za treniranje i validaciju koji su dobiveni metodom unakrsne validacije.

UNAKRSNA VALIDACIJA/PROVJERA (*eng. Cross validation*)

Unakrsna validacija (*eng. LOO-„leaving-one-out“)* je jednostavna tehnika za ocjenu točnosti klasifikacijskog modela. Skup podataka se razdijeli u *k* podskupova približno istih veličina. U jednoj iteraciji (*k-1)* podskupova se koristi za učenje modela koji se potom testira na preostalom testnom podskupu. Postupak se ponavlja *k* puta, sve dok se svaki podskup ne nađe jednom u ulozi testnog podskupa. Ova metoda testirana je na različitom broju podskupova te se kao najbolje rješenje, pogotovo za velike skupove, nametnula validacija s 10 podskupova (n=10).

U ovom radu koristit će se unakrsnu validaciju s tri podskupa (n=3). Razlog tome je što na svaki klasifikator treba primijeniti ovu metodu. S obzirom da postoje 4 klasifikatora, a svaki treba provući tri puta kroz RF mijenjajući pritom podskupove za učenje i validaciju, ukupno će se slučajne šume pozvati 15 puta (4 klasifikatora x 3 + ulazni decimirani skup podataka x 3).

Program Slucajne sume.m koristi se u postupku pripremanja podataka za rad sa Slučajnim šumama. On podijeli podatke svakog Klasifikatora u tri jednake skupine. Prvi puta će se gornja skupina koristiti za validaciju, a druge dvije za učenje. Drugi puta će se srednja skupina koristiti za validaciju, a druge dvije za učenje. Treći puta se donja skupina koristi za validaciju, a druge dvije za učenje. Sljedeće slika ilustrirati način pripreme podataka za RF.



Slika 12: Priprema podataka za Slučajne šume korištenjem unakrsne validacije

4.6. REZULTATI DOBIVENI SLUČAJNIM ŠUMAMA

Statistički paket Slučajne šume formiraju tri izlazne datoteke: glasovi.txt, izlmatrica.txt i vaznost.txt.

Svaki skup podataka jednog klasifikatora je podijeljen u tri dijela. RF je ispitivao svaki podskup i za njega stvorio datoteku glasovi.txt. U toj datoteci se nalaze tri stupca. Prvi stupac označava redni broj stabla, drugi udio stabala koja su glasovala protiv u ukupnom broju i treći udio stabala koja su glasovala za. Sve glasove jednog klasifikatora potrebno je spojiti u jednu datoteku. Podaci se upisuju jedni ispod drugih pomoću programa Glasovi.m. Nove stvorene datoteke ujedinjenih glasova svakog klasifikatora te ulaznog skupa podataka upotrebljavaju se za crtanje Precision-Recall krivulje. Opis programa Precision-Recall krivulja.m:

- Čitanjem datoteke glasovi.txt klasifikatora 1 izračunava se postotak točnih glasova (6) koji se upisuje u prvi stupac matrice. Na analogni način dobivaju se i preostala tri stupca.

$$Broj\ točnih = \frac{za}{(za + protiv)} \qquad (6)$$

- Čitanjem redaka datoteke („*C_elegans_tocni+decimirani_netocni_acc.txt*“) u šesti stupac matrice upisuje se vrijednost 0 ili 1 ovisno o točnosti mjesta izrezivanja. Ako je na kraju pročitanog reda slovo „F“, mjesto izrezivanja je netočno pa će se u zadnji stupac matrice upisati 0, ako je mjesto izrezivanja bilo točno u zadnji stupac matrice upisuje se 1.

- Prag se linearno povećava od 0 do 1. Peti stupac nastaje uspoređivanjem praga s prvih četiri stupaca. Ako je bilo koji od tih stupaca veći od praga (ili funkcija) u matricu se upisuje 1 dok za slučaj kada niti jedan nije veći od praga se upisuje 0.

Precision-Recall krivulja crta se na temelju četiri parametra:

- TP – True positives → detektor (odluka RF-a) je rekao da je točno i bilo je točno (odluka donesena na temelju ulaznih podataka)
- TN – True negatives → detektor kaže da je netočno i njegova odluka je prema ulaznim podacima točna
- FP – False positives → detektor kaže da je točno i njegova odluka je prema ulaznim podacima netočna (bilo je netočno)
- FN – False negatives → detektor kaže da je netočno i njegova odluka je prema ulaznim podacima netočna (bilo je točno)

Formule za izračunavanje varijabli Precision i Recall su:

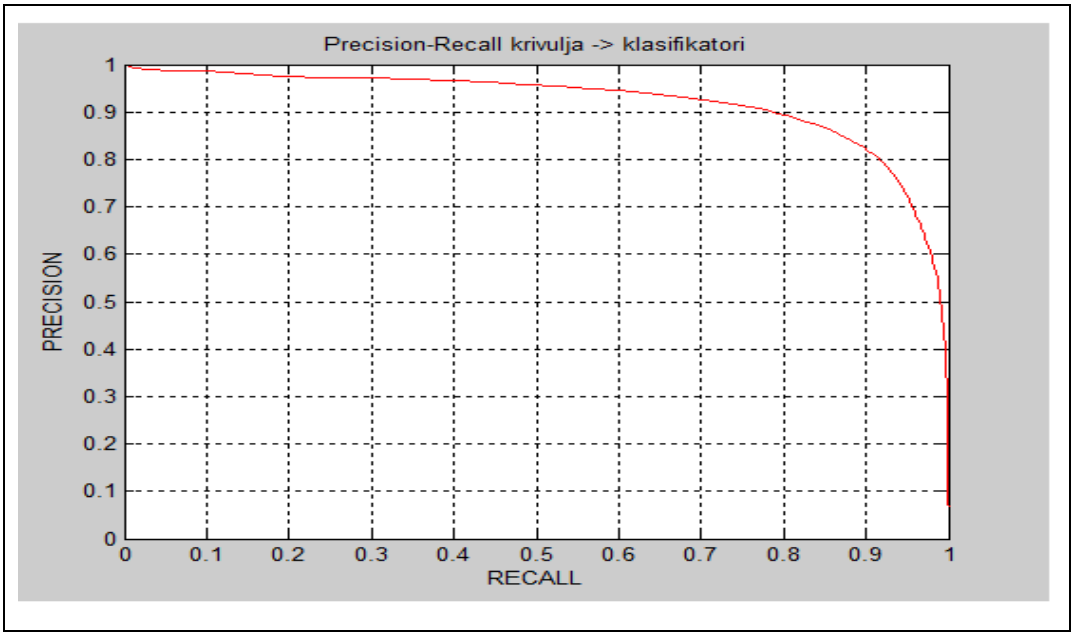
$$Precision = \frac{TP}{TP+FP} \qquad (7) \qquad\qquad Recall = \frac{TP}{TP+FN} \qquad (8)$$

Svakom parametru dodijeljene su kombinacije binarnih znakova dobivene usporedbom podataka dobivenih detektorom i točnih ulaznih podataka.

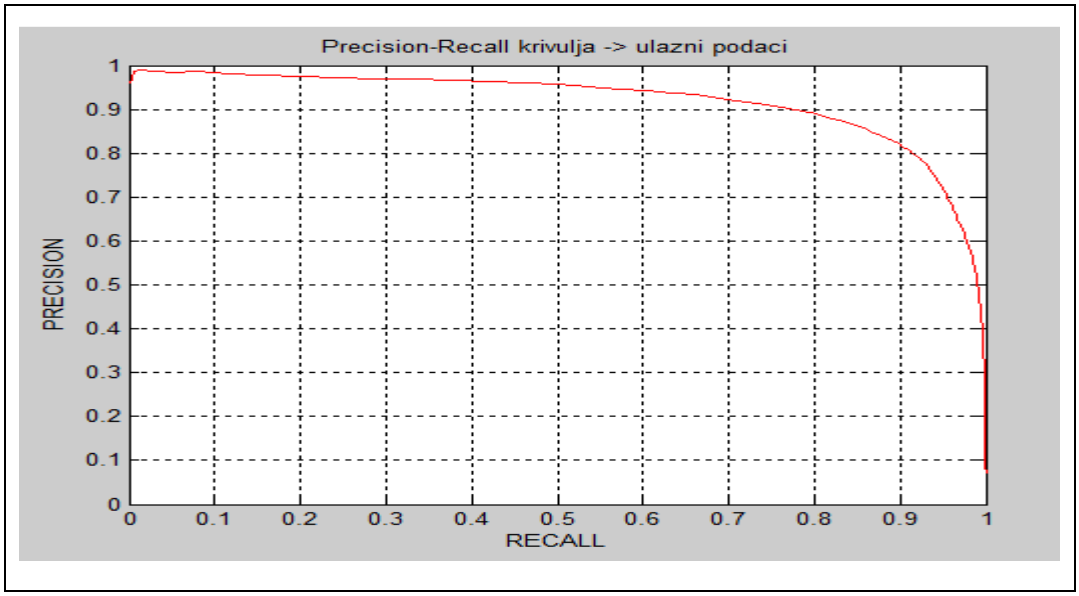
- TP = '11'
- FP = '10'
- FN = '01'
- TN = '00'

Precision i Recall računaju se za različite pragove koji se linearno mijenjaju u intervalu [0,1] s pomakom 0.01. Svakom pragu odgovara jedna točka krivulje.

Precision – Recall krivulje dobivene na temelju podataka obrađenih iz RF-a



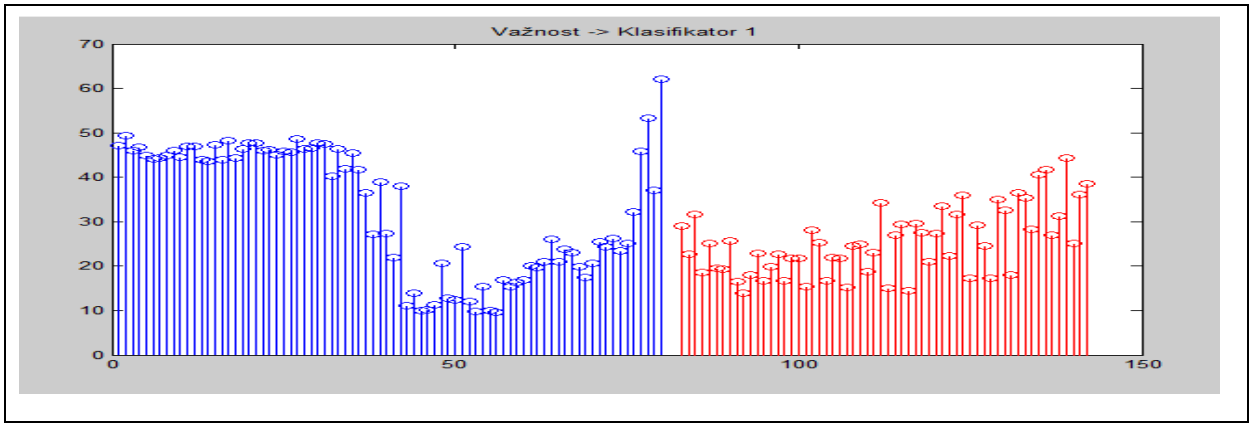
Slika 13: Precision-Recall krivulja dobivena na temelju podjele ulaznog skupa podataka na klasifikatore



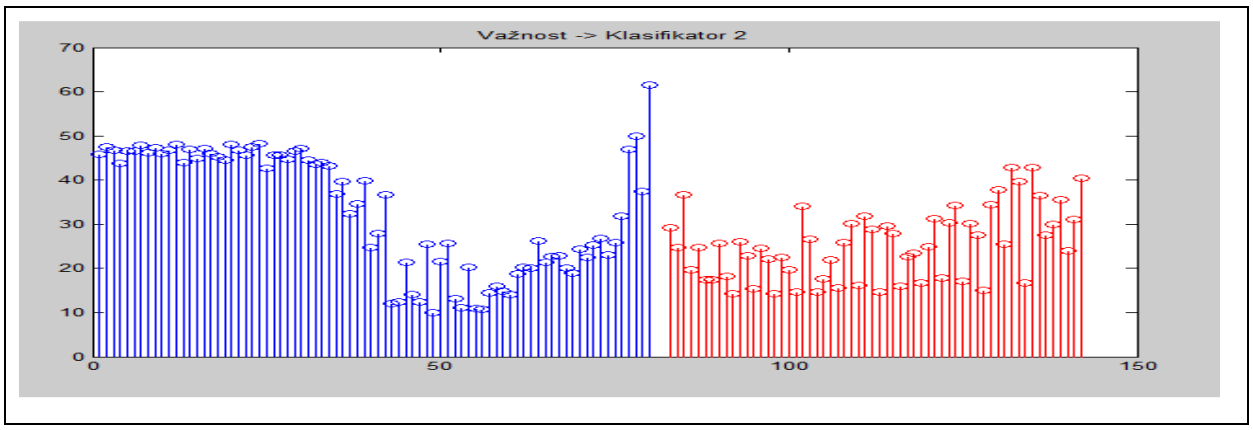
Slika 14: Precision-Recall krivulja dobivena na temelju ulaznih podataka

Uspoređivanjem Precision-Recall krivulja dobivenih na temelju glasova klasifikatora i ulaznih podataka (svi točni i decimirani netočni), obje rađene metodom unakrsne validacije, jako teško je odlučiti koja je bolja. Stoga možemo zaključiti da ovim načinom podjele ulaznih podataka u klasifikatore nismo značajnije doprinijeli poboljšanju krivulje odnosno detektora za otkrivanje točnih mjesta izrezivanja.

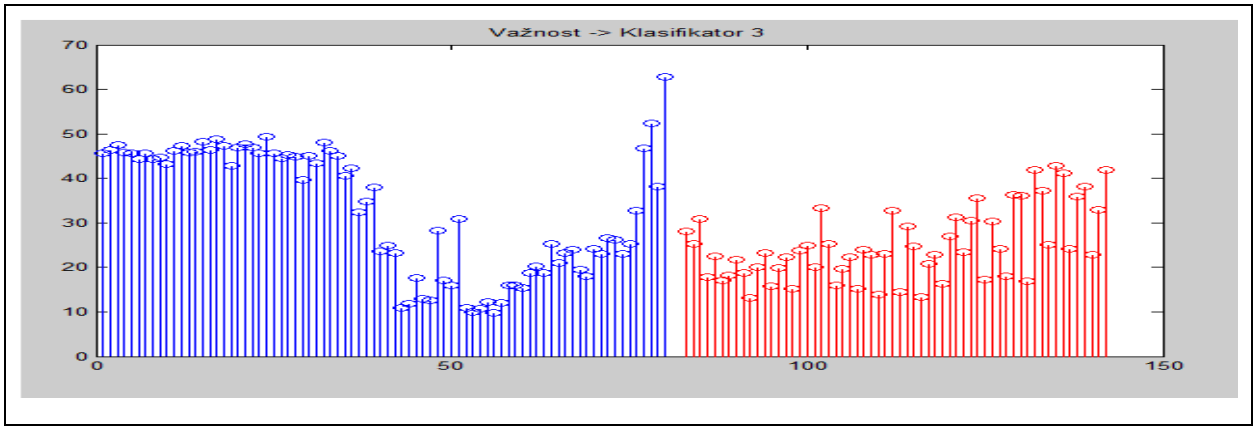
Datoteka važnost određuje važnost pojedinih pozicija u blizini akceptorskog mjesta izrezivanja. S obzirom da su za svaki klasifikator stvorene tri datoteke važnosti, usrednjavanjem će se dobiti srednja vrijednost važnosti pojedine pozicije svakog klasifikatora. Program kojim se crta graf važnosti naziva se Vaznost.m.



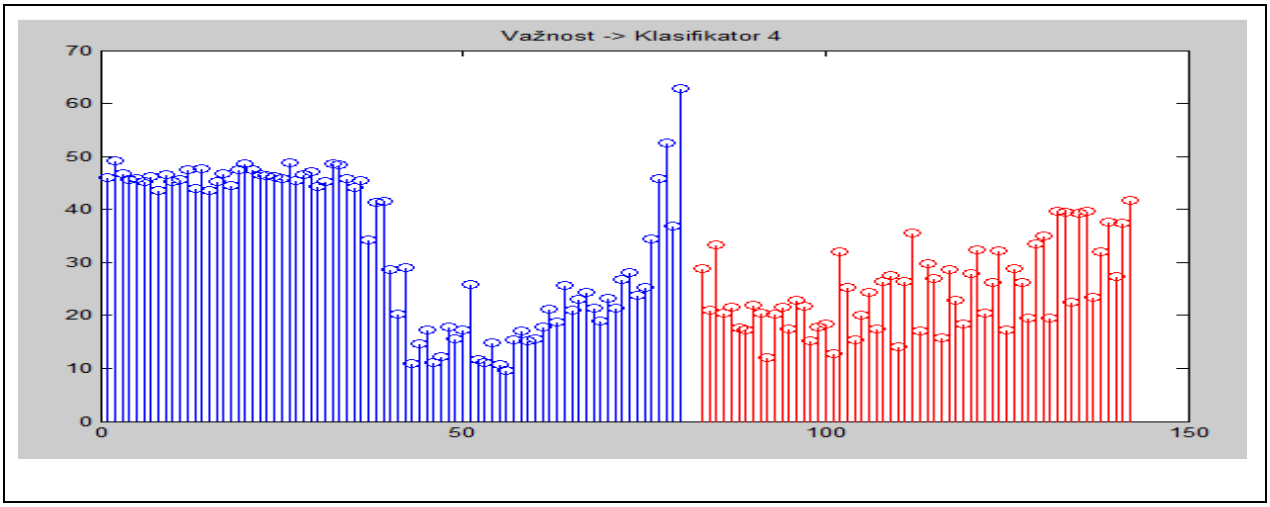
Slika 15: Važnost Klasifikatora 1



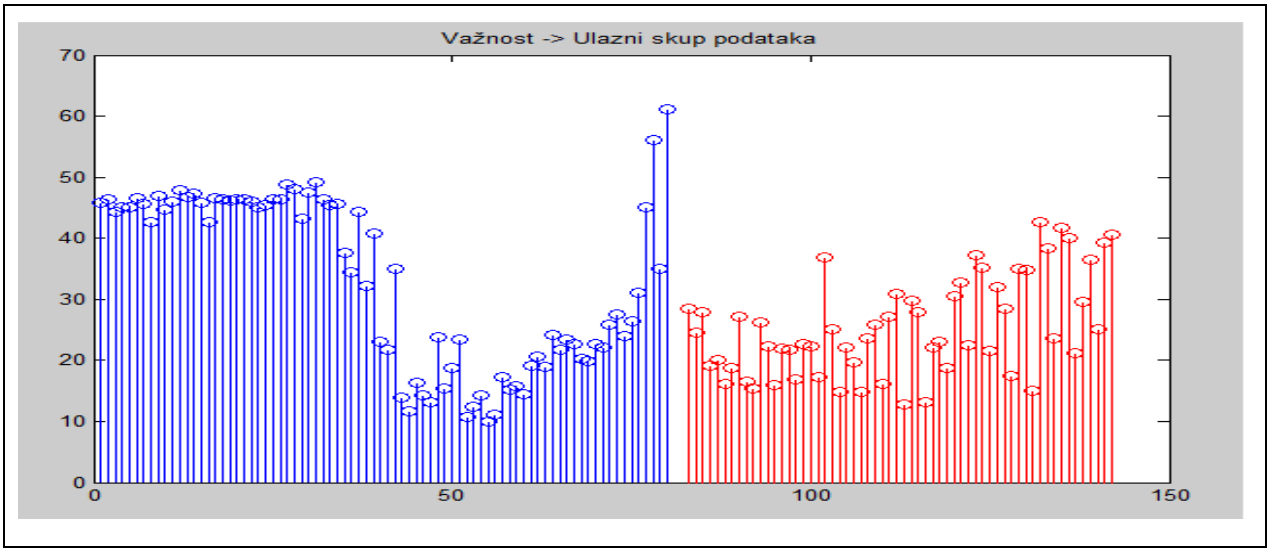
Slika 16: Važnost Klasifikatora 2



Slika 17: Važnost Klasifikatora 3



Slika 18:Važnost Klasifikatora 4



Slika 19:Važnost dobivena na temelju ulaznih podataka

Na temelju grafa važnosti može potvrditi teoriju o velikom značaju prvih pet slova ispred akceptorskog mjesta izrezivanja. Također možemo vidjeti da nukleotidi nešto dalje od samog mjesta izrezivanja imaju veći značaj neko nukleotidi uz samo mjesto izrezivanja. Uspoređivanje grafova svih klasifikatora možemo zaključiti se oni međusobno slažu o važnosti pojedinih mjesta i da niti jedan od njih se posebno ne ističe neka druga mjesta.

5. ZAKLJUČAK

Ovim radom pokušalo se primijeniti znanja iz vektorske kvantizacije na problem određivanja točnih mjesta izrezivanja introna. Slijed od sedam nukleotida (pet prije akceptorskog mjesta + AG) prebačen je u višedimenzionalni prostor. Na temelju Linde-Buzo-Gray algoritma te koljena distorzije pokazalo se da postoji četiri načina točnih mjesta izrezivanja. Zbog veličine originalnog skupa podataka, ulazni skup je smanjen tako da su uzimana sva točna mjesta izrezivanja i svako deseto netočno. Ulazni skup podataka upisuje se u četiri klasifikatora, zbog činjenice da se točna mjesta izrezivanja mogu podijeliti u četiri skupine. Svaki klasifikator sadrži sva netočna mjesta izrezivanja te samo ona točna koja pripadaju jednoj skupini dok se ostala točna mjesta deklariraju kao netočna. Na dobivenim klasifikatorima upotrebljava se statistička metoda Slučajne šume, a izlazne vrijednoste prikazuju se pomoću Precision-Recall krivulja i grafova važnosti.

Uspoređivanjem Precision-Recall krivulja dobivenih na temelju glasova klasifikatora i ulaznih podataka jako je teško odlučiti koja je bolje. Stoga možemo zaključiti da ovim načinom podjele ulaznih podataka nismo značajniji doprinijeli poboljšanju krivulje odnosno detektora za otkrivanje točnih mjesta izrezivanja. Također na temelju ovog rada nije moguće donijeti odluku o broju stvarnih načina točnih mjesta izrezivanja. Mogući uzroci zbog kojih Precision-Recall krivulja nije ispala bolje su: premali broj nukleotida na kojima se koristio algoritam te korištenje euklidske udaljenosti u višedimenzionalnom prostoru.

Uspoređivanjem grafova važnosti svih klasifikatora možemo potvrditi teoriju o velikom značaju prvih pet slova ispred akceptorskog mjesta izrezivanja. Također možemo primijetiti kako se svi klasifikatori međusobno slažu oko važnosti pojedinih mjesta te da niti jedan ne ističe neko drugo mjesto. Na temelju grafova važnosti možemo zaključiti da bi se sljedeća istraživanja trebala bazirati na mjestima koja se ne nalaze uz samo mjesto izrezivanja, jer su upravo to mjesta koja su svi klasifikatori označili kao važnima.

6. LITERATURA

[1] Parađina N., Analiza DNK metodama obrade signala u vremenskoj i frekvencijskoj domeni, Diplomski rad br. 1143, FER, 2008.

[2] Penović M., diplomski rad, 2008.

[3] Sören Sonnenburg, Gabriele Schweikert, Petra Philips, Jonas Behr, Gunnar Rätsch, *Accurate splice site prediction using support vector machines,* http://www.biomedcentral.com/content/pdf/1471-2105-8-S10-S7.pdf, 1.6.2009.

[4] Matej Marić, Iva Milošević, Silvano Perković, Mihovil Viduka, Skalarna kvantizacija signala

[5] Tomislav Puđa, Predviđanje otapalu dostupnog područja površine proteina, rujan 2008.

[6] http://en.wikipedia.org/wiki/Precision_and_recall, 5.6.2009.

7. SAŽETAK/ ABSTRACT

Klasifikacija kodirajućih regija u genomu

Danas korištene metode za detekciju mjesta izrezivanja introna su vrlo uspješne na jednostavnijim organizmima, dok im je uspješnost znatno lošija na složenijima. U ovom radu pokušala su se iskoristiti znanja iz vektorske kvantizacije za određivanje broja načina točnih mjesta izrezivanja. Dobiveni klasifikatori korišteni su za treniranje i testiranje statističkog paketa slučajne šume. Izlazne vrijednosti prikazane su Precision-Recall krivuljom te grafom važnosti. Iz krivulja dobivenih na temelju glasova klasifikatora i ulaznih podataka teško je zaključiti koja je bolja. Zaključak je da ovim načinom podjele ulaznih podataka nismo značajnije doprinijeli poboljšanju krivulje odnosno detektora za otkrivanje točnih mjesta izrezivanja. Također na temelju ovog rada nije moguće donijeti odluku o broju stvarnih načina točnih mjesta izrezivanja.

Ključne riječi: gen, genom, ekson, intron, detektor, nukleotid, kodirajući, nekodirajući, mjesto izrezivanja, podjela u grupe, unakrsna validacija

Classification of coding regions in the genome

Today, methods used to detect the splice sites of introns are very successful in simpler organisms, but their performance is significantly worse in more complex. In this paper we tried to use knowledge from the vector quantization to determine the exact number of ways to accurate splice-site a intron. Obtained clasifiers were used to train and test statistical package random forest. Output values are shown in Precision-Recall curves, and in the importance graph. From the curves obtained on the basis of votes classifiers and input data, it is difficult to conclude which is better. The conclusion is that this way of division of input data did not significantly contribute to the improvement of the curve or detector that can detect the accurate splice-site of a intron. Also on the basis of this work is not possible to make a decision on the number of real ways to accurate splice-site an intron.

Words: gene, genome. exon, intron, detector, nucleotide, coding, non-coding, splice-site, clustering, cross validation.

PRIVITAK

➔ Klaster.m

```
%% Postupak dobivanja koljena ditorzije
% Zelimo pomocu Linde-Buzo-Gray algoritma otkriti u koliko ce nakupina
(oblaka)
% algoritam sortirati ulazne podatke (u ovom slucaju samo tocne)

%% Podatke slazemo u matricu
fid=fopen('C_elegans_samo_tocni_acc.txt','r');
for n=1:401
    data = fgets(fid);
end

matrica_podataka(224,398) = char(0);
k=1;
for k=1:Inf
    data = fgets(fid);
    if isnumeric(data), break, end % end of the file
        for i=1:2:length(data)
            matrica_podataka(k,round(i/2))=data(i);
        end
end

% matrica_podataka(:,399)=[]-> brisanje zadnjeg znaka, ako je prazan

% Close file
fclose(fid);

%% 4 moguca slova, udaljenosti medu njima moraju biti jednake - 3D
prostor
% A=[1 0 0], C=[0 1 0], G=[0 0 1], T=[-1/3 -1/3 -1/3]

% Ucitavamo vektor po vektor matrice, rasprusujemo podatke
n=193, m=1, i=1, p=1;
while (n<=199)
    while (m<k)
         if (matrica_podataka(m,n)=='A')
             x(i,p)=[1];
             y(i,p)=[0];
             z(i,p)=[0];
         end

         if (matrica_podataka(m,n)=='C')
             x(i,p)=[0];
             y(i,p)=[1];
             z(i,p)=[0];
         end

         if (matrica_podataka(m,n)=='G')
             x(i,p)=[0];
             y(i,p)=[0];
             z(i,p)=[1];
         end

         if (matrica_podataka(m,n)=='T')
             x(i,p)=[-1/3];
```

```
            y(i,p)=[-1/3];
            z(i,p)=[-1/3];
        end
        m=m+1;
        i=i+1;
    end
    m=1;
    i=1;
    p=p+1;
    n=n+1;
end

%% Glavne varijable
X = [x, y, z];   % Matrica podataka.

i=1, k=10
while (i<=k)
    Muk = i+1;             % Broj centara
    wx = ones(size(X));    % Vektor tezina

  % Pomocne varijable
    ep_nut = 1e-4; ep_fs = 1e-5;  % Relativni stop kriteriji
    pertub =0;                    % Perturbacija potrebna za povecanje
broja kvanata
    sum_or_avg = 0;
    bemin = 2;                    % Najmanji broj vektora u jednoj grupi
    debug = 1;                    % Ako zelimo kontrolne ispise

  % Poziv funkcije za vektorsku kvantizaciju
   [centri, da, brojVektora, distorzije, pripadnost,in]= ...
      lbg_wed_nut(X,wx,Muk,ep_nut,sum_or_avg,pertub,ep_fs,bemin,debug);

  %% Objasnjenje i pregled izlaznih varijabli
  % Centri koje je pronasao algoritam
    centri

  % U pojednim grupama (clusters) ima odredjeni broj vektora - neki su
vjerojatno "zalutali"
    brojVektora

  % Po svakoj grupi imamo drugacije rasprsenje i distorziju (srednju
kvadratnu gresku)
    distorzije

    d(i)=sum(distorzije)
    i=i+1
end

subplot(2,1,1), plot([2:i],d), grid, xlabel('Distorzija')
d=log(d)
subplot(2,1,2), plot([2:i],d), grid, xlabel('Distorzija u logaritamskom
mjerilu')

% Matrica u koju spremamo nagibe distorzije
matrica=0;
[m,n]=size(matrica);

while (m<k)
    n=1
```

```
    matrica(m,n)=[d(m)]
    n=2
    matrica(m,n)=[d(m+1)]
    m=m+1
end

% Radimo omjere susjenih nagiba distorzije
r=1
while (r<k-1)
    T(r)=matrica(r,:)/matrica(r+1,:)
    r=r+1
end

% Dobivene susjedne omjere oduzimamo i usporedujemo - Najveca razlika
(koljeno distorzije)
r=1
razlika=T(r)-T(r+1)
najveci=razlika

r=2, a=1
while (r<k-3)
    razlika=T(r)-T(r+1)
    if (razlika>najveci)
        najveci=razlika
        a=r
    end
    r=r+1
end

% Nasli smo broj centara
i=a;
Muk = i+2;           % Broj centara
wx = ones(size(X));  % Vektor tezina

% Pomocne varijable
ep_nut = 1e-4; ep_fs = 1e-5;  % Relativni stop kriteriji
pertub =0;                % Perturbacija potrebna za povecanje broja
kvanata
sum_or_avg = 0;
bemin = 2;           % Najmanji broj vektora u jednoj grupi
debug = 1;           % Ako zelimo kontrolne ispise

% Poziv funkcije za vektorsku kvantizaciju
[centri, da, brojVektora, distorzije, pripadnost,in]= ...
   lbg_wed_nut(X,wx,Muk,ep_nut,sum_or_avg,pertub,ep_fs,bemin,debug);
```

➔ **Razvrstavanje.m**

```
%% Razvrstavanje u vrece
% Iz koljena distorzije mozemo zakljuciti da se tocna mjesta izrezivanja
% grupiraju u 4 oblaka (Muk=4)
% Sljedeci korak je razvrstati tocna mjesta izrezivanja u vrece ovisno o
% tome kojem oblaku pripadaju

% broj redaka pripadnosti ili X oznacavamo s k
% broj centara odnosno broj oblaka oznacavamo s i
% in - indeks matrica_podataka grupirani po clusterima - prvi cluster (
svi
% indeksi ulazne matrice koji pripadaju prvom clusteru iduci prema
rastucim
% predznacima), drugi cluster (svi indeksi ulazne matrice drugog
clustera)...
% pripadnost - oznacava koji redak x pripada kojem centru

%% Stvaramo 4 datoteke
fid1= fopen('prva_vreca_acc.txt','w');
fid2= fopen('druga_vreca_acc.txt','w');
fid3= fopen('treca_vreca_acc.txt','w');
fid4= fopen('cetvrta_vreca_acc.txt','w');

k=64838
% Ucitavamo u varijabele kolko ima koji cluster elemenata
for(i=1:4)
    if(i==1)
        a=brojVektora(i);
    end
    if(i==2)
        b=brojVektora(i);
    end
    if(i==3)
        c=brojVektora(i);
    end
    if (i==4)
        d=brojVektora(i);
    end
end

prva_vreca=zeros(a,398); druga_vreca=zeros(b,398);
treca_vreca=zeros(c,398); cetvrta_vreca=zeros(d,398);
j=1; u=1; v=1; z=1; t=1;

for(i=1:k)
    j=in(i);
    if (pripadnost(j)==1)
        prva_vreca(u,:)=matrica_podataka(j,:);
        u=u+1;
    end
    if (pripadnost(j)==2)
        druga_vreca(v,:)=matrica_podataka(j,:);
        v=v+1;
    end

    if (pripadnost(j)==3)
        treca_vreca(z,:)=matrica_podataka(j,:);
        z=z+1;
    end
```

```
    if (pripadnost(j)==4)
        cetvrta_vreca(t,:)=matrica_podataka(j,:);
        t=t+1;
    end
end

% Upis prve_vrece u datoteku s headerom
fprintf(fid1, '@relation prva_vreca_acc.txt\n');
for n=1:398
    fprintf(fid1, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid1, '@attribute class {F,T}\n');
fprintf(fid1, '\n');
fprintf(fid1, '@data\n');

for(i=1:a)
    data=prva_vreca(i,:);
    fprintf(fid1, '%c,%c,%c,……%c,%c,%c,', data([1:398]));
    tocnost= 'T';
    fprintf(fid1, '%c\n', tocnost);
end

% Upis druge_vrece u datoteku s headerom
fprintf(fid2, '@relation druga_vreca_acc.txt\n');
for n=1:398
    fprintf(fid2, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid2, '@attribute class {F,T}\n');
fprintf(fid2, '\n');
fprintf(fid2, '@data\n');

for(i=1:b)
    data=druga_vreca(i,:);
    fprintf(fid2, '%c,%c,%c,……%c,%c,%c,', data([1:398]));
    tocnost= 'T';
    fprintf(fid2, '%c\n', tocnost);
end

% Upis trece_vrece u datoteku s headerom
fprintf(fid3, '@relation treca_vreca_acc.txt\n');
for n=1:398
    fprintf(fid3, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid3, '@attribute class {F,T}\n');
fprintf(fid3, '\n');
fprintf(fid3, '@data\n');

for(i=1:c)
    data=treca_vreca(i,:);
    fprintf(fid3, '%c,%c,%c,……%c,%c,%c,', data([1:398]));
    tocnost= 'T';
    fprintf(fid3, '%c\n', tocnost);
end

% Upis cetvrte_vrece u datoteku s headerom
fprintf(fid4, '@relation cetvrta_vreca_acc.txt\n');
for n=1:398
    fprintf(fid4, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid4, '@attribute class {F,T}\n');
```

```
fprintf(fid4, '\n');
fprintf(fid4, '@data\n');

for(i=1:d)
    data=cetvrta_vreca(i,:);
    fprintf(fid4, '%c,%c,%c,……%c,%c,%c,', data([1:398]));
    tocnost= 'T';
    fprintf(fid4, '%c\n', tocnost);
end

% Close file
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);

%% U sljedece matrice upisujemo po 7 slova iz svakog retka pojedine vrece
% Tih 7 slova najbitniji su u procesu izrezivanja introna
prva_vreca_slova=prva_vreca(:,193:199);
druga_vreca_slova=druga_vreca(:,193:199);
treca_vreca_slova=treca_vreca(:,193:199);
cetvrta_vreca_slova=cetvrta_vreca(:,193:199);
```

➔ **Tocni_netocni(decimirani).m**

```
%% Postupak decimiranja ulaznog skupa podataka
%% Citanje ulazne datoteke i sortiranje podataka u novoj

fid1=fopen('C_elegans_tocni+netocni_acc.txt','r');
fid2=fopen('C_elegans_tocni+decimirani_netocni_acc.txt','w');

% Priprema headera nove datoteke za rad sa slucajne sume
fprintf(fid2, '@relation C_elegans_tocni+decimirani_netocni_acc.txt\n');

for n=1:142
    fprintf(fid2, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid2, '@attribute class {F,T}\n');
fprintf(fid2, '\n');
fprintf(fid2, '@data\n');

% Citanje ulaznog skupa podataka
for n=1:402
    data = fgets(fid1);
end
k=1;
for(k=1:Inf)
data = fgets(fid1);
    if isnumeric(data), break, end
    % Uzmi svaki deseti netocni
    if (data(797)=='F')
        if (rand(1,1)>0.9)
        fprintf(fid2, '%c%c%c……c%c%c', data([395-160:397+120+1]));
        fprintf(fid2, '%c\n', 'F');
        end
    end
    % Ako je tocan provjeri u kojoj se vreci nalazi i to zapisi na kraj
tog reda
    if (data(797)=='T')
        matrica=data(385:2:397);

        % Trazi tocne u prvoj vreci -> a=9867 br. redaka u prvoj vreci
        i=1;
        while (i<=9867)
            s=0;j=1;
            while(j<=7 && s~=1) % trazi dok je isto
                if (matrica(1,j)~=char(prva_vreca_slova(i,j)))
                    s=1;   % ako je razlicito
                else
                    s=0;   % ako je isto
                end
                j=j+1;
            end
            i=i+1;
        end
            if(s==0)
            fprintf(fid2, '%c%c%c……c%c%c', data([395-160:397+120+1]));
            fprintf(fid2, '%c,%c\n', 'T1');
            end
```

```
 % Trazi tocne u drugoj vreci ->b=31405
        i=1;
        while (i<=31045 && s==1) %ako je razlicito, trazis u drugoj vreci
            s=0;j=1;
            while(j<=7 && s~=1)
                if (matrica(1,j)~=char(druga_vreca_slova(i,j)))
                    s=1;
                else
                    s=0;
                end
                j=j+1;
            end
            i=i+1;
        end
            if(s==0)
            fprintf(fid2, '%c%c%c……%c%c%c', data([395-160:397+120+1]));
            fprintf(fid2, '%c,%c\n', 'T2');
            end

        % Trazi tocne u trecoj vreci -> c=13935
        i=1;
        while (i<=13935 && s==1)
            s=0;j=1;
            while(j<=7 && s~=1)
                if (matrica(1,j)~=char(treca_vreca_slova(i,j)))
                    s=1;
                else
                    s=0;
                end
                j=j+1;
            end
            i=i+1;
        end
            if(s==0)
            fprintf(fid2, '%c%c%c……%c%c%c', data([395-160:397+120+1]));
            fprintf(fid2, '%c,%c\n', 'T3');
            end

          % Trazi tocne u cetvrtoj vreci -> d=9631
            i=1;
        while (i<=9631 && s==1)
            s=0;j=1;
            while(j<=7 && s~=1)
                if (matrica(1,j)~=char(cetvrta_vreca_slova(i,j)))
                    s=1;
                else
                    s=0;
                end
                j=j+1;
            end
            i=i+1;
        end
            if(s==0)
            fprintf(fid2, '%c%c%c…….c%c%c', data([395-160:397+120+1]));
            fprintf(fid2, '%c,%c\n', 'T4');
            end
    end
end
fclose(fid1);
fclose(fid2);
```

➔ **Klasifikatori.m**

```
%% Postupak sortiranja ulaznog decimiranog skupa podataka u Klasifikatore

% Svaki Klasifikator se dobiva tako da se netocni podaci prepisuju, a kod
% tocnih se ispituje zadnje slovo u svakom retku koje oznacava kojem
% klasifikatoru pripada to mjesto izrezivanja
% npr. za klasifikator 1 ce sva mjesta izrezivanja koja imaju na kraju
% oznaku 1 biti spremljena kao tocna, a sva mjesta izrezivanja koja imaju
% oznaku nekog drugog klasifikatora biti spremljena kao netocna

%% Citanje ulaznog skupa podataka i stvaranje datoteka klasifikatora

fid=fopen('C_elegans_tocni+decimirani_netocni_acc.txt','r');
fid1=fopen('Klasifikator1_acc.txt','w');
fid2=fopen('Klasifikator2_acc.txt','w');
fid3=fopen('Klasifikator3_acc.txt','w');
fid4=fopen('Klasifikator4_acc.txt','w');

% header Klasifikatora 1
fprintf(fid1, '@relation Klasifikator1_acc.txt\n');

for n=1:142
    fprintf(fid1, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid1, '@attribute class {F,T}\n');
fprintf(fid1, '\n');
fprintf(fid1, '@data\n');

% header Klasifikatora 2
fprintf(fid2, '@relation Klasifikator2_acc.txt\n');

for n=1:142
    fprintf(fid2, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid2, '@attribute class {F,T}\n');
fprintf(fid2, '\n');
fprintf(fid2, '@data\n');

% header Klasifikatora 3
fprintf(fid3, '@relation Klasifikator3_acc.txt\n');

for n=1:142
    fprintf(fid3, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid3, '@attribute class {F,T}\n');
fprintf(fid3, '\n');
fprintf(fid3, '@data\n');

% header Klasifikatora 4
fprintf(fid4, '@relation Klasifikator4_acc.txt\n');

for n=1:142
    fprintf(fid4, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid4, '@attribute class {F,T}\n');
fprintf(fid4, '\n');
fprintf(fid4, '@data\n');
```

```
% Ucitavamo red po red iz ulazne datoteke
for n=1:146
    data = fgets(fid);
end
k=1;
for(k=1:Inf)
data = fgets(fid);
if isnumeric(data), break, end
% Ako je netocno, samo prepisujemo u svaki Klasifikator
if (data(285)=='F')
    fprintf(fid1,'%c%c%c……%c%c%c\n', data([1:285]));
    fprintf(fid2,'%c%c%c……%c%c%c\n', data([1:285]));
    fprintf(fid3,'%c%c%c……%c%c%c\n', data([1:285]));
    fprintf(fid4,'%c%c%c……%c%c%c\n', data([1:285]));
end
% Ako je tocno, ispitujemo zadnje slovo
if(data(285)=='T')
    % Pripada Klasifikatoru 1
    % Tamo se sprema taj red kao tocan u sve druge klasifikatore kao
netocan
    if(data(287)=='1')
    fprintf(fid1,'%c%c%c%……%c%c%c\n', data([1:285]));
    data(285)=='F';
    fprintf(fid2,'%c%c%c%……%c%c%c\n', data([1:285]));
    fprintf(fid3,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    fprintf(fid4,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    end
    % Pripada Klasifikatoru 2
    if(data(287)=='2')
    fprintf(fid2,'%c%c%c%……%c%c%c\n', data([1:285]));
    data(285)=='F';
    fprintf(fid1,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    fprintf(fid3,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    fprintf(fid4,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    end
    % Pripada Klasifikatoru 3
    if(data(287)=='3')
    fprintf(fid3,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    data(285)=='F';
    fprintf(fid1,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    fprintf(fid2,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    fprintf(fid4,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    end
    % Pripada Klasifikatoru 4
    if(data(287)=='4')
    fprintf(fid4,'%c%c%c%……%c%c%c%c\n', data([1:285]));
    data(285)=='F';
    fprintf(fid1,'%c%c%c%……%c%c%c\n', data([1:285]));
    fprintf(fid2,'%c%c%c%……%c%c%c\n', data([1:285]));
    fprintf(fid3,'%c%c%c%……%c%c%c\n', data([1:285]));
    end
 end
end
fclose(fid);
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
```

➔ **Slucajne sume.m**

```
%% Postupak pripremanja podataka za rad sa statistickom metodom Random
Forest
% Podaci svakog Klasifikatora se podijele u tri jednake skupine
% Prvi puta će gornja skupina biti za test, a druge dvije za testiranje
% Drugi puta ce srednja biti za test, druge dvije za testiranje
% Treci puta ce donja biti za test, a druge dvije za testiranje
% Ova metoda raspodjele podataka i takvog provodenja testiranja se naziva
Cross validation

fid=fopen('Klasifikator1_acc.txt','r');

fid1=fopen('Klasifikator1_1_test_acc.txt','w'); % 1 test, 2 i 3 training
fid2=fopen('Klasifikator1_1_training_acc.txt','w');
fid3=fopen('Klasifikator1_2_test_acc.txt','w'); % 2 test, 1 i 3 training
fid4=fopen('Klasifikator1_2_training_acc.txt','w');
fid5=fopen('Klasifikator1_3_test_acc.txt','w'); % 3 test, 1 i 2 training
fid6=fopen('Klasifikator1_3_training_acc.txt','w');

% header Klasifikatora1_1_test
fprintf(fid1, '@relation Klasifikator1_1_test_acc.txt\n');

for n=1:142
    fprintf(fid1, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid1, '@attribute class {F,T}\n');
fprintf(fid1, '\n');
fprintf(fid1, '@data\n');

% header Klasifikatora1_1_training
fprintf(fid2, '@relation Klasifikator1_1_training_acc.txt\n');

for n=1:142
    fprintf(fid2, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid2, '@attribute class {F,T}\n');
fprintf(fid2, '\n');
fprintf(fid2, '@data\n');

% header Klasifikatora1_2_test
fprintf(fid3, '@relation Klasifikator1_2_test_acc.txt\n');

for n=1:142
    fprintf(fid3, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid3, '@attribute class {F,T}\n');
fprintf(fid3, '\n');
fprintf(fid3, '@data\n');

% header Klasifikatora1_2_training
fprintf(fid4, '@relation Klasifikator1_2_training_acc.txt\n');

for n=1:142
    fprintf(fid4, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid4, '@attribute class {F,T}\n');
fprintf(fid4, '\n');
fprintf(fid4, '@data\n');
```

```
% header Klasifikatora1_3_test
fprintf(fid5, '@relation Klasifikator1_3_test_acc.txt\n');

for n=1:142
    fprintf(fid5, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid5, '@attribute class {F,T}\n');
fprintf(fid5, '\n');
fprintf(fid5, '@data\n');

% header Klasifikatora1_3_training
fprintf(fid6, '@relation Klasifikator1_3_training_acc.txt\n');

for n=1:142
    fprintf(fid6, '@attribute NA%d {A,T,G,C}\n', n);
end
fprintf(fid6, '@attribute class {F,T}\n');
fprintf(fid6, '\n');
fprintf(fid6, '@data\n');

% citanje ulazne datoteke i razvrstavanje podataka
for n=1:146
    data = fgets(fid);
end
k=1;broj=103570  %k=310704/3
for(k=1:Inf)
    data = fgets(fid);
    if isnumeric(data), break, end
    if (0<k && k<=broj)
        fprintf(fid1,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid4,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid6,'%c%c%c……%c%c%c%c\n', data([1:285]));
    end
    if(broj<k && k<=2*broj)
        fprintf(fid2,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid3,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid6,'%c%c%c……%c%c%c\n', data([1:285]));
    end
    if(2*broj<k && k<=3*broj)
        fprintf(fid2,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid4,'%c%c%c……%c%c%c\n', data([1:285]));
        fprintf(fid5,'%c%c%c……%c%c%c\n', data([1:285]));
    end
end


fclose(fid);
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
fclose(fid5);
fclose(fid6);
```

➔ **Precision-Recall krivulja**

```
%% PRECISION - RECALL KRIVULJA

% Open file
  fid1=fopen('glasovi_Klasifikator_1.txt','r');
  fid2=fopen('glasovi_Klasifikator_2.txt','r');
  fid3=fopen('glasovi_Klasifikator_3.txt','r');
  fid4=fopen('glasovi_Klasifikator_4.txt','r');

  fid5 = fopen('C_elegans_tocni+decimirani_netocni_acc.txt', 'r');

% upis prvih cetri stupca matrice
  matrica=zeros(900000, 6);
  for k=1:Inf
        s=fgets(fid1);
        if isnumeric(s), break, end
        i=14;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
             protiv=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
             protiv=str2num(s(i:i+6));
        else protiv=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        i=24;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
            za=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
            za=str2num(s(i:i+6));
        else za=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        postotak=za/(protiv+za); % postotak tocnih
        matrica(k,1)=postotak;
  end

   for k=1:Inf
        s=fgets(fid2);
        if isnumeric(s), break, end
        i=14;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
             protiv=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
             protiv=str2num(s(i:i+6));
        else protiv=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        i=24;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
            za=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
            za=str2num(s(i:i+6));
        else za=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        postotak=za/(protiv+za);
        matrica(k,2)=postotak;
   end

   for k=1:Inf
        s=fgets(fid3);
        if isnumeric(s), break, end
        i=14;
```

```
        if (s(i)~=char(32)) % broj ima 8 znamenaka
             protiv=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
             protiv=str2num(s(i:i+6));
        else protiv=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        i=24;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
            za=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
            za=str2num(s(i:i+6));
        else za=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        postotak=za/(protiv+za);
        matrica(k,3)=postotak;
    end

    for k=1:Inf
        s=fgets(fid4);
        if isnumeric(s), break, end
        i=14;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
             protiv=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
             protiv=str2num(s(i:i+6));
        else protiv=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        i=24;
        if (s(i)~=char(32)) % broj ima 8 znamenaka
            za=str2num(s(i:i+7));
        elseif(s(i+1)~=char(32))  % broj ima 7 znamenaka
            za=str2num(s(i:i+6));
        else za=str2num(s(i:i+5)); %broj ima 6 znamenaka
        end
        postotak=za/(protiv+za);
        matrica(k,4)=postotak;
   end

% kontrolna točka
    kt={'Prva cetri stupca upisana!'}

% upis 6. stupca matrice - iz originalne datoteke
    for n=1:146
     data = fgets(fid5);
    end

    i=1;
    for k=1:Inf
            data = fgets(fid5);
            if isnumeric(data), break, end
            if(data(285)=='T')
                matrica(i,6)=1;
            else
                matrica(i,6)=0;
            end
            i=i+1;
    end

% kontrolna točka
    kt={'Upisan je sesti stupac!'}
```

```
% upis 5. stupac matrice
   br_redova=k-1;
   prag=0; precision=zeros(1,100); recall=zeros(1,100);
   for br=1:100
       sumTP=0; sumTN=0; sumFP=0; sumFN=0;

       for i=1:br_redova
            if (prag<matrica(i,1) || prag<matrica(i,2) ||
prag<matrica(i,3) || prag<matrica(i,4))
                matrica(i,5)=1;
            else
                matrica(i,5)=0;
            end

            if(matrica(i,5)==1 && matrica(i,6)==1)
               sumTP=sumTP+1;
            end

            if(matrica(i,5)==0 && matrica(i,6)==1)
                sumFN=sumFN+1;
            end

            if(matrica(i,5)==1 && matrica(i,6)==0)
                sumFP=sumFP+1;
            end

            if(matrica(i,5)==0 && matrica(i,6)==0)
                sumTN=sumTN+1;
            end
       end
   precision(br)=sumTP/(sumTP+sumFP*10);
%precision(br)=sumTP/(sumTP+10*sumFP)-desetkovani;
   recall(br)=sumTP/(sumTP+sumFN);
   prag=prag+0.01;
end

%kontrolna točka
 kt={'Upisani su peti stupac!'}

% crtanje precision-recall krivulje
figure,grid, plot(recall,precision,'r')
xlabel('RECALL');
ylabel('PRECISION');

fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
fclose(fid5);
```