Klasifikacija PMBC ćelija

Seminarski rad u okviru kursa
Istraživanje podataka 2
Matematički fakultet

Marko Veljković 43/2015

mi15043@alas.matf.bg.ac.rs

Sadržaj

1 Uvod

Ulazne datoteke sadrže podatke dobijene iz perifernih mononukleoznih krvnih ćelija (eng. *PBMCs*). PBMC ćelije uključuju ćelije različitih tipova: limfocite, monocite i dendritske ćelije koje se koriste u istraživanju u različitim oblastima biomedicine, uključujući infektivne bolesti, imunologiju, malignitet, transplantacionu imunologiju, razvoj vakcina, i skrining. Mada mogu imati različite funkcije, glavna funkcija PBMC ćelija je imuna odbrana organizma. Svaki tip ćelije ima karakteristične obrasce proteina i gena koje ih međusobno razlikuju i mogu da se koriste za podelu prema njihovom tipu.

Datoteke 029_pluripotent_stem_cell-derived_SCGB3A2+_airway_epithelium_csv i 030_pluripotent_stem_cell-derived_SCGB3A2+_airway_epithelium_csv sadrže podatke vezane za ekspresiju 31221 gena na skupu od redom 742 i 714 PBMC ćelija. Skup gena koji su posmatrani je identičan u obe datoteke i uređen po istom redosledu. Nazivi svih gena imaju prefiks hg38 koji označava da su podaci vezani za 38. verziju referentnog humanog genoma.

Sadržaj datoteka predstavlja pluripotentne[1] matične ćelije, tj. ćelije koje grade epitelno tkivo disajnih puteva. U prvoj datoteci se nalaze podaci o matičnim ćelijama u gornjim disajnim putevima, dok su u drugoj podaci o matičnim ćelijama donjih disajnih puteva.

Ulazna datoteka sadrži podatke u CSV formatu u obliku ukrštene tabele. Svaka datoteka ima 31222 reda, pri čemu prvi red sadrži redni broj ćelije za koju je vršeno istraživanje, a prva kolona sadrži identifikaciju gena. Polje u preseku prvog reda i prve kolone je inicijalno prazno, a mi ćemo ubaciti ime 'geni' radi lakšeg rukovanja sa imenima gena. Vrednosti u matrici (koje ne pripadaju prvom redu/koloni - 313221xN vrednosti) sadrže broj transkripta gena (navedenog u prvoj koloni) u ćeliji.

Tema u nastavku rada će biti učitavanje pomenutih datoteka [2.1] pretprocesiranje podataka [2.2] konstruisanje različitih modela klasifikacije [3] koji će biti napravljeni tako da podatke klasifikuju u jednu od dve klase, jedna klasa će predstavljati prvu datoteku, a druga klasa drugu datoteku, kako jednostavnih [3.1] tako i složenijih korišćenjem ansambl tehnika [3.2]. Na kraju ćemo izvršiti detaljnu analizu dobijenih rezultata kao i međusobno poređenje svake metode posebno [4], svih jednostavnih metoda [4.8] kao i rezultate svih ansambl tehnika [4.9].

[1] Postojanje neprepoznatih matičnih ćelija pokazano je indirektnim zapažanjima, a da nije utvrđen tačan izgled i građa tih ćelija. Pluripotentne matične ćelije su jedna od podvrsta neprepoznatih matičnih ćelija

2 Analiza i pretprocesiranje podataka

2.1 Učitavanje i pregled podataka

Da bismo mogli da učitamo podatke, prvo moramo uključiti biblioteku za rad sa csv datotekama.

```
import pandas as pd
```

Kod 1: Uključivanje biblioteke 'pandas'

```
first_file = '029_pluripotent_stem_cell-derived_SCGB3A2+_airway_epithelium_csv.csv'
second_file = '030_pluripotent_stem_cell-derived_SCGB3A2+_airway_epithelium_csv.csv'
df_first = pd.read_csv(first_file).set_index('geni')
df_second = pd.read_csv(second_file).set_index('geni')
```

Kod 2: Učitavanje podataka

Nakon učitavanja podataka, transponujemo obe matrice i time dobijamo matrice u kojima geni predstavljaju kolone, a ćelije redove.

```
# dfft i dfst sadrze transponovane podatke iz tabela df_first i df_second
dfft = df_first.T
dfst = df_second.T
```

Kod 3: Transponovanje tabela

Zatim obema matricama dodajemo po jednu kolonu koja će predstavljati klasu podataka, elementi prve datoteke će pripadati klasi 1, a elementi druge datoteke klasi 2.

```
dfft['class'] = 1
dfst['class'] = 2
```

Kod 4: Dodavanje kolone 'class'

Sada želimo da proverimo i ispišemo na standardni izlaz koliko redova i kolona sadrže matrice.

```
print("Dimenzija podataka prve datoteke: " + str(dfft.shape))
print("Dimenzija podataka druge datoteke: " + str(dfst.shape) + '\n')
```

Kod 5: Ispis dimenzija matrica

```
Dimenzija podataka prve datoteke: (742, 31222)
Dimenzija podataka druge datoteke: (714, 31222)
```

Slika 1: Dimenzije ulaznih podataka

Na kraju želimo da prikažemo deo podataka iz obe matrice, nad kojima ćemo vršiti pretprocesiranje i klasifikaciju.

```
print("Deo podataka prve datoteke:\n" + str(dfft.head(20)))
print("Deo podataka druge datoteke:\n" + str(dfst.head(20)) + '\n')
```

Kod 6: Ispis dela podataka iz obe datoteke

Deo podataka prve datoteke:

geni	hg38_A1BG	hg38_A1BG-AS1	hg38_A1CF	...	hg38_ZYX	hg38_ZZEF1	class
1	0	0	1	...	1	0	1
2	0	0	0	...	0	0	1
3	0	0	6	...	10	1	1
4	0	0	0	...	0	0	1
5	0	0	2	...	0	0	1
6	0	0	0	...	0	0	1
7	0	0	0	...	0	0	1
8	0	1	2	...	5	0	1
9	0	0	12	...	3	0	1
10	1	0	0	...	1	0	1
11	0	0	0	...	1	0	1
12	0	0	2	...	4	2	1
13	0	0	4	...	1	1	1
14	0	0	5	...	1	0	1
15	0	0	0	...	0	2	1
16	0	0	0	...	0	0	1
17	1	0	0	...	0	0	1
18	0	0	0	...	0	0	1
19	1	1	5	...	6	0	1
20	0	0	0	...	0	0	1

Slika 2: Podaci iz prve datoteke

Deo podataka druge datoteke:

geni	hg38_A1BG	hg38_A1BG-AS1	hg38_A1CF	...	hg38_ZYX	hg38_ZZEF1	class
1	0	0	0	...	1	0	2
2	0	0	0	...	0	0	2
3	1	0	0	...	0	1	2
4	0	0	0	...	0	0	2
5	0	0	0	...	0	0	2
6	1	0	0	...	1	0	2
7	2	0	0	...	2	0	2
8	1	0	0	...	1	0	2
9	0	0	0	...	0	0	2
10	0	0	0	...	1	1	2
11	0	0	0	...	0	0	2
12	1	0	0	...	0	0	2
13	2	0	0	...	0	0	2
14	1	0	0	...	2	0	2
15	0	0	0	...	0	0	2
16	0	0	0	...	0	0	2
17	1	0	0	...	1	0	2
18	0	0	0	...	0	0	2
19	3	0	0	...	1	0	2
20	0	0	0	...	1	0	2

Slika 3: Podaci iz druge datoteke

2.2 Pretprocesiranje

Pre početka klasifikacije, moramo obraditi i pripremiti sirove podatke za rad sa njima. Korišćene metode su provera nepostojećih vrednosti, uklanjanje nula-redova i nula-kolona, detekcija i izdvajanje elemenata van granica.

Spajamo matrice podataka u jednu matricu nad kojom ćemo vršiti pretprocesiranje.

```
dft = pd.concat([dfft, dfst])
```

Kod 7: Spajanje matrica

Funkcija 'concat' spaja prosleđene matrice po istoimenim kolonama, što su u našem slučaju nazivi gena. Ovime smo dobili jednu matricu koja sadrži sve podatke učitane na početku.

2.2.1 Provera nepostojećih vrednosti

```
# Provera da li skup podataka sadrzi neku NaN vrednost
if dft.isnull().values.any():
    print("Skup sadrzi NaN vrednosti!\n")
else:
    print("Skup ne sadrzi nijednu NaN vrednost\n")
```

Kod 8: Provera postojanja NaN vrednosti

Izvršavanjem uslovne naredbe dobijamo ispis 'Skup ne sadrži nijednu NaN vrednost'.

2.2.2 Uklanjanje nula-redova i nula-kolona

Razlog uklanjanja nula-redova i nula-kolona je što takvi redovi, odnosno kolone nisu značajni za generisanje modela klasifikacije, a njihovim uklanjanjem podižemo efikasnost izvršavanja programa.

```
def remove_zero(df):
    izbaceni = df[df.sum(axis=1) == 0]
    df.drop(df[df.sum(axis=1) == 0].index, inplace=True)
    return df, izbaceni
```

Kod 9: Funkcija koja uklanja nula-redove ili nula-kolone

Kada ovu funkciju pozovemo sa parametrom 'dft' ulanjamo nula-redove, a kada je pozovemo sa 'dft.T', odnosno transponovanom matricom 'dft', funkcija uklanja nula-kolone. Nakon pozivanja funkcije nad transponovanom matricom, moramo transponovati matricu još jednom, kako bi je vratili u pređašnje stanje.

```
dft, zero_rows = remove_zero(dft)
df, zero_cols = remove_zero(dft.T)
dft = df.T
```

Kod 10: Uklanjanje nula-redova i nula-kolona

```
print("Dimenzija podataka nakon uklanjanja 0 redova: " + str(dft.shape))
print("Dimenzija podataka nakon uklanjanja 0 kolona: " + str(dft.shape))
```

Kod 11: Ispis dimenzije matrice nakon uklanjanja nula-redova i nula-kolona

```
Dimenzija podataka nakon uklanjanja 0 redova: (1456, 31222)
Dimenzija podataka nakon uklanjanja 0 kolona: (1456, 21275)
```

Slika 4: Dimenzije podataka nakon uklanjanja nula-redova i nola-kolona

Vršimo proveru dimenzije matrice, kako bismo utvrdili koliko nula redova i kolona je bilo u objedinjenoj matrici, čime dobijamo podatak da nije bilo nula-redova, a otkrili smo i uklonili 9947 nula-kolona.

2.2.3 Detekcija i izdvajanje elemenata van granice

Sledeći korak u pretprocesiranju podataka je otkrivanje i izdvajanje elemenata van granica, odnosno elemenata čije se vrednosti značajno razlikuju od vrednosti ostalih podataka u posmatranom skupu. Razlog izdvajanja je njihov moguć uticaj na generisanje modela za klasifikaciju, čime bi doprineli da model ne funkcioniše na najbolji mogući način.

U nastavku je data implementacija funkcije, koja korišćenjem Z-vrednosti, pronalazi i izdvaja elemente van granica iz trenutnog skupa podataka, a istovremeno razdvaja elemente van granica na one koji se nalaze ispod donje i iznad gornje granice. Razlog ovom razdvajanju je mogućnost postojanja bitnih podataka u elementima iznad gornje granice i njih želimo posebno obraditi u daljem radu.

Z-vrednost svakog podatka X_{ij} se računa po formuli $Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j}$, gde X_{ij} predstavlja vrednost trenutno posmatranog elementa (i-ti red, j-ta kolona), μ_j srednju vrednost j-te kolone, a σ_j standardnu devijaciju elemenata j-te kolone.

```
from scipy import stats
def detect_and_exclude_outliers(df):
    outliers_small = df[(stats.zscore(df) <= -3).any(axis=1)]
    outliers_big = df[(stats.zscore(df) >= 3).any(axis=1)]
    df_without_outliers = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
    return df_without_outliers, outliers_big, outliers_small
```

Kod 12: Funkcija za detekciju i izdvajanje elemenata van granica

Želimo da izdvojimo elemente van granica i pozivamo funkciju 'detect_and_exclude_outliers'.

```
df, outliers_big, outliers_small = detect_and_exclude_outliers(dft)
```

Kod 13: Pozivanje funckije za detekciju i izdvajanje elemenata van granica

Nakon njenog izvršavanja ispisujemo dimenzije novodobijenih matrica.

```
print("Dimenzija podataka nakon uklanjanja elemenata van granica: " + str(dft.shape))
print("Dimenzija elemenata van donje granice: " + str(outliers_small.shape))
print("Dimenzija elemenata van gornje granice: " + str(outliers_big.shape) + "\n")
```

Kod 14: Ispis dimenzija matrica dobijenih izvršavanjem funcije 'detect_and_exclude_outliers'

```
Dimenzija podataka nakon uklanjanja elemenata van granica: (0, 21275)
Dimenzija elemenata van donje granice: (0, 21275)
Dimenzija elemenata van gornje granice: (1456, 21275)
```

Slika 5: Dimenzije matrica dobijenih izvršavanjem funcije 'detect_and_exclude_outliers' nad redovima

Možemo primetiti da su svi elementi proglašeni za elemente iznad gornje granice što nas dovodi do zaključka da svi redovi sadrže neku vrednost atributa koja ih odvaja od ostalih redova, tako da ovu tehniku izdvajanja elemenata van granica ne možemo koristiti. Sledeća stvar koju ćemo pokušati je provera da li postoje redovi koji se po svim atributima izdvajaju od ostalih. Da bismo ovo proverili, prvo moramo izmeniti funkciju ’detect_and_exclude_outliers’.

```
from scipy import stats
def detect_and_exclude_outliers(df):
    outliers_small = df[(stats.zscore(df) <= -3).all(axis=1)]
    outliers_big = df[(stats.zscore(df) >= 3).all(axis=1)]
    df_without_outliers = df[(np.abs(stats.zscore(df)) < 3).any(axis=1)]
    return df_without_outliers, outliers_big, outliers_small
```

Kod 14: Izmenjena funkcija za detekciju i izdvajanje elemenata van granica

Ponovo ispisujemo dobijene rezultate na standardni izlaz.

```
Dimenzija podataka nakon uklanjanja elemenata van granica:(1456, 21275)
Dimenzija elemenata van donje granice: (0, 21275)
Dimenzija elemenata van gornje granice: (0, 21275)
```

Slika 6: Dimenzije matrica dobijenih izvršavanjem funcije 'detect_and_exclude_outliers' nad redovima

Dobili smo da nijedan red podataka nije okarakterisan kao element van granice, pa nećemo koristiti ni ovaj način izdvajanja elemenata van granice. Pokušaćemo malo drugačiji pristup, umesto da posmatramo redove, pokušaćemo da izdvojimo kolone koje predstavljaju elemente van granica, ali pre toga implementaciju funkcije ’detect_and_exclude_outliers’ ponovo menjamo, tako da je ona ista kao što je prikazano u [Kod 12].

Prilikom pozivanja funkcije šaljemo transponovanu matricu, tako da nakon dobijenih rezultata, sve tri novodobijene matrice (originalnu matricu bez elemenata van granica, matricu elemenata ispod donje granice i matricu elemenata iznad gornje granice) moramo ponovo transponovati.

```
df, outliers_big, outliers_small = detect_and_exclude_outliers(dft.T)
dft = df.T
outliers_small = outliers_small.T
outliers_big = outliers_big.T
```

Kod 14: Pozivanje funckije za detekciju i izdvajanje elemenata van granica

Kao i do sada, nakon pozivanja funkcije ispisujemo dobijene rezultate.

```
Dimenzija podataka nakon uklanjanja elemenata van granica: (1456, 20712)
Dimenzija elemenata van donje granice: (1456, 0)
Dimenzija elemenata van gornje granice: (1456, 563)
```

Slika 7: Dimenzije matrica dobijenih izvršavanjem funcije 'detect_and_exclude_outliers'

Utvrdili smo da postoje 563 gena koji predstavljaju elemente van granica, njih smo izdvojili iz originalnog skupa podataka(skup podataka koji predstavlja krajnji rezultat pretprocesiranja ulaznih podataka, sada i bez elemenata van granica), i smestili u posebnu promenljivu ‘outliers_big’. Kako smo uvideli da ne postoje elementi ispod donje granice, sve elemente van granice smeštamo u promenljivu ‘outliers’.

U toku izdvajanja elemenata van granica, izgubili smo informaciju o klasi kojoj oni pripadaju, pa moramo ponovo eksplicitno navesti, kako bismo mogli kasnije izvršiti njihovo klasifikovanje. Ovo možemo uraditi pošto znamo da matrica sa originalnim podacima i matrica sa elementima van granica imaju isti broj redova.

```
outliers = outliers_big
class_column = dft.values[:, -1:]
outliers['class'] = class_column
```

Kod 15: Dodavanje oznake klase u matricu elemenata van granica

3 Klasifikacija

Sada kada smo pripremili podatke, možemo početi sa generisanjem modela za klasifikaciju. Skup podataka delimo na 2 dela, prvi predstavlja matricu sa svim podacima bez kolone 'class', ovu matricu ćemo u nastavku zvati originalni podaci, a drugi vektor-kolonu koja sadrži oznake klasa kojima pripadaju redovi matrice. Isto ovo radimo i za matricu u kojoj se nalazi podaci koji predstavljaju elemente iznad gornje granice, u nastavku elementi van granice.

```
# X predstavlja podatke za klasifikaciju
x = dft.values[:, :-1]
# ox predstavlja elemente van granica za klasifikaciju
ox = outliers.values[:, :-1]
# Y predstavlja klase kojima podaci pripadaju
y = dft.values[:, -1:]
# oy predstavlja klase kojima elementi van granica pripadaju,
# medjutim kako je taj podatak jednak koloni y,
# samo cemo izjednaciti ove dve promenljive
oy = y
```

Kod 16: Podela matrice na podatke za klasifikaciju i vektor-kolonu sa oznakama klasa

Sada želimo dobijene podatke da podelimo na trening i test skup, a za to nam je potrebna 'train_test_split' funkcija iz 'sklearn' biblioteke.

```
from sklearn.model_selection import train_test_split
```

Kod 17: Uključivanje 'train_test_split' funkcije

Matricu sa podacima i vektor-kolonu šaljemo 'train_test_split' funkciji, koja deli matricu podataka na trening i test skup. Kao treći parametar se prosleđuje broj u opsegu [0,1], koji označava deo podataka koji će biti izdvojen za testiranje, u ovom slučaju to je 0.3.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
ox_train, ox_test, oy_train, oy_test = train_test_split(ox, oy, test_size=0.3)
```

Kod 18: Podela podataka na trening i test skup

Za klasifikaciju podataka korišćene su:

- Jednostavne metode
  - K najbližih suseda
  - Drvo odlučivanja
  - Mašine sa potpornim vektorima
- Ansambl tehnike[2]
  - Nasumična šuma (eng. Random forest)
  - Pakovanje (eng. Bagging)
  - Pojačavanje (eng. Boosting)
  - Glasanje (eng. Voting)

[2] Tehnike koje kombinuju rezultate nekoliko, ne nužno različitih, modela dobijenih nad trening podacima

Dve funkcije koje ćemo koristiti u svim metodama, radi lakšeg testiranja modela i ispisivanja dobijenih rezultata su ‘fit_model’ i ‘prediction’.

```
def fit_model(model, x_train, x_test, y_train, y_test, mode, method):
    if mode == 'save':
        model.fit(x_train, y_train.ravel())
        dump(model, os.path.join('models', method + '.joblib'))
    print("Rezultat trening skupa: " + str(model.score(x_train, y_train)))
    print("Rezultat test skupa: " + str(model.score(x_test, y_test)))
```

Kod 19: Definicija funkcije 'fit_model'

```
from sklearn.metrics import confusion_matrix
def prediction(model, x_train, x_test, y_train, y_test):
    y_train_predicted = model.predict(x_train)
    y_test_predicted = model.predict(x_test)
    print("Matrica kofuzije trening skupa:\n" + str(confusion_matrix(y_train, y_train_predicted)))
    print("Matrica kofuzije test skupa:\n" + str(confusion_matrix(y_test, y_test_predicted)))
```

Kod 20: Definicija funkcije 'prediction'

Funkcija ‘fit_model’ proverava da li je vrednost parametra ‘mode’ jednak “save” i ukoliko jeste, pravi novi model, koji trenira na osnovu prosleđenih trening podataka, koji nakon završenog treniranja, čuva dobijeni model na relativnoj putanji ‘./models/ime_metoda.joblib’. Kada zapamti model na pomenutoj putanji ili ukoliko vrednost parametra ‘mode’ nije bila jednaka “save”, ispisuje rezultat primene modela, na trening i test skup, na standardni izlaz.

Funkcija ‘prediction’ na osnovu prosleđenog parametra ‘model’, predviđa kojoj klasi pripada svaki od redova u trening i test skupu, te dobijene rezultate upisuje u promenljive ‘y_train_predicted’ i ‘y_test _predicted’. Na osnovu dobijenih rezultata pravi matrice konfuzije za oba skupa i dobijene rezultate ispisuje na standardni izlaz.

Još jedna funkcija koja je korišćena u navedenim funkcijama je ‘os.path.join’, služi za spajanje prosleđenih putanja pomoću sistemskog separatora direktorijuma. Da bi smo koristi ovu funkciju moramo uključiti biblioteku ‘os’.

```
import os
```

Kod 21: Uključivanje biblioteke 'os'

Pre nego što počnemo sa klasifikacijom, moramo navesti još jednu biblioteku koju ćemo koristiti u nastavku, a to je ‘time’ biblioteka, koja će nam poslužiti za merenje vremena izvršavanja svih algoritama pojedinačno.

```
import time
```

Kod 22: Učitavanje biblioteke 'time'

Sve funkcije koje će biti navedene u nastavku, imaju jedan zajednički parametar ‘mode’, koji ukoliko ima vrednost “load”, označava da se model učitava sa prosleđene putanje, dok se u suprotnom pravi novi i trenira nad trening skupom.

3.1 Jednostavne klasifikacione metode

3.1.1 K najbližih suseda

Osnovna ideja je da na osnovu k najbližih suseda datog sloga odredimo kojoj klasi on pripada. Vrši se određivanje najbližih suseda, zatim prebrojavanje koliko suseda pripada kojoj klasi. Ona klasa kojoj pripada najviše suseda je dodeljena posmatranom slogu.

```
from sklearn.neighbors import KNeighborsClassifier
def knn(x_train, x_test, y_train, y_test, non, mode, rtype):
    print("-----K najblizih suseda:-----")
    start = time.time()
    if mode == 'load':
        model = load(os.path.join('models', 'knn_' + rtype + '.joblib'))
    else:
        model = KNeighborsClassifier(n_neighbors=non, weights='uniform')
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'knn_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print("Vreme izvrsavanja: " + str(time.time()-start) + '\n')
```

Kod 23: Definicija funkcije k najbližih suseda

Parametar 'non' označava broj suseda koji se prosleđuje klasifikatoru 'KNeighborsClassifier', mi ćemo testirati sa 3, 5 i 10 suseda.

```
knn(x_train, x_test, y_train, y_test, non=3, mode='save', rtype='gen')
knn(x_train, x_test, y_train, y_test, non=5, mode='save', rtype='gen')
knn(x_train, x_test, y_train, y_test, non=10, mode='save', rtype='gen')
knn(ox_train, ox_test, oy_train, oy_test, non=3, mode='save', rtype='out')
knn(ox_train, ox_test, oy_train, oy_test, non=5, mode='save', rtype='out')
knn(ox_train, ox_test, oy_train, oy_test, non=10, mode='save', rtype='out')
```

Kod 24: Pozivanja funkcije knn nad originalnim podacima

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.979
Rezultat test skupa: 0.977
Matrica kofuzije trening skupa:
[[500  20]
 [  1 498]]
Matrica kofuzije test skupa:
[[213   9]
 [  1 214]]
Vreme izvrsavanja: 107.122
```

Slika 8: Knn metoda sa 3 suseda nad originalnim podacima

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.968
Rezultat test skupa: 0.974
Matrica kofuzije trening skupa:
[[489  31]
 [  1 498]]
Matrica kofuzije test skupa:
[[212  10]
 [  1 214]]
Vreme izvrsavanja: 105.178
```

Slika 9: Knn metoda sa 5 suseda nad originalnim podacima

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.965
Rezultat test skupa: 0.967
Matrica kofuzije trening skupa:
[[486  34]
 [  1 498]]
Matrica kofuzije test skupa:
[[208  14]
 [  0 215]]
Vreme izvrsavanja: 106.348
```

Slika 10: Knn metoda sa 10 suseda nad originalnim podacima

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.984
Rezultat test skupa: 0.941
Matrica kofuzije trening skupa:
[[475  14]
 [  2 528]]
Matrica kofuzije test skupa:
[[229  24]
 [  2 182]]
Vreme izvrsavanja: 1.707
```

Slika 11: Knn metoda sa 3 suseda nad elementima van granice

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.975
Rezultat test skupa: 0.920
Matrica kofuzije trening skupa:
[[469  20]
 [  5 525]]
Matrica kofuzije test skupa:
[[220  33]
 [  2 182]]
Vreme izvrsavanja: 2.064
```

Slika 12: Knn metoda sa 5 suseda nad elementima van granice

```
-----K najblizih suseda:-----
Rezultat trening skupa: 0.970
Rezultat test skupa: 0.927
Matrica kofuzije trening skupa:
[[466  23]
 [  8 522]]
Matrica kofuzije test skupa:
[[223  30]
 [  2 182]]
Vreme izvrsavanja: 2.227
```

Slika 13: Knn metoda sa 10 suseda nad elementima van granice

3.1.2 Drvo odlučivanja

Problem klasifikacije rešavamo postavljanjem pitanja o vrednostima atributa podataka iz trening skupa. Svaki put kada dobijemo odgovor postavljamo novo pitanje, dok ne dođemo do zaključka o klasi posmatranog sloga. Klasifikator 'DecisionTreeClassifier' u python3 je implementiran korišćenjem CART (Classification And Regression Trees) algoritma. Mi ćemo koristiti samo rešenje klasifikacionog problema algoritma, koji predviđa vrednost kategoričke klase na osnovu neprekidnih i/ili kategoričkih atributa. Ukoliko se ne navede drugačije, kao meru nečistoće uzimamo Ginijev indeks.

```
from sklearn.tree import DecisionTreeClassifier
def dtc(x_train, x_test, y_train, y_test, mode, rtype, criteria, depth):
    print("-----Drvo odlucivanja:-----")
    start = time.time()
    if mode == 'load':
        model = load(os.path.join('models', 'dtc_' + rtype + '.joblib'))
    else:
        model = DecisionTreeClassifier(criterion=criteria, max_depth=depth)
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'dtc_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 25: Definicija funkcije drveta odlučivanja

Funkciji šaljemo trening skup, test skup i jednu od dve mere nečistoće koje ćemo koristiti za pravljenje modela (Ginijev indeks i entropiju). U prvom testiranju nećemo ograničavati dubinu drveta.

```
dtc(x_train, x_test, y_train, y_test, mode='save', rtype='gen', criteria='gini', depth=None)
dtc(x_train, x_test, y_train, y_test, mode='save', rtype='gen', criteria='entropy', depth=None)
dtc(ox_train, ox_test, oy_train, oy_test, mode='save', rtype='out', criteria='gini', depth=None)
dtc(ox_train, ox_test, oy_train, oy_test, mode='save', rtype='out', criteria='entropy', depth=None)
```

Kod 26: Pozivanje funkcije drveta odlučivanja bez ograničavanja dubine

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.970
Matrica kofuzije trening skupa:
[[541   0]
 [  0 478]]
Matrica kofuzije test skupa:
[[196   5]
 [  8 228]]
Vreme izvrsavanja: 1.787
```

Slika 14: Dtc metoda, originalni podaci, ginijev indeks, bez ograničenja dubine

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.970
Matrica kofuzije trening skupa:
[[541   0]
 [  0 478]]
Matrica kofuzije test skupa:
[[195   6]
 [  7 229]]
Vreme izvrsavanja: 1.851
```

Slika 15: Dtc metoda, originalni podaci, entropija, bez ograničenja dubine

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[528   0]
 [  0 491]]
Matrica kofuzije test skupa:
[[214   0]
 [  2 221]]
Vreme izvrsavanja: 0.063
```

Slika 16: Dtc metoda, elementi van granice, ginijev indeks, bez ograničenja dubine

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.998
Matrica kofuzije trening skupa:
[[528   0]
 [  0 491]]
Matrica kofuzije test skupa:
[[214   0]
 [  1 222]]
Vreme izvrsavanja: 0.109
```

Slika 17: Dtc metoda, elementi van granice, entropija, bez ograničenja dubine

Na osnovu dobijenih rezultata, možemo zaključiti da prilikom klasifikovanja originalnih podataka, dolazi do preprilagođavanja podacima. Zato ćemo sada ponovo pozvati funkciju, uz dodatno ograničavanje dubine stabla, do petog naslednika.

```
dtc(x_train, x_test, y_train, y_test, mode='save', rtype='gen', criteria='gini', depth=5)
dtc(x_train, x_test, y_train, y_test, mode='save', rtype='gen', criteria='entropy', depth=5)
dtc(ox_train, ox_test, oy_train, oy_test, mode='save', rtype='out', criteria='gini', depth=5)
dtc(ox_train, ox_test, oy_train, oy_test, mode='save', rtype='out', criteria='entropy', depth=5)
```

Kod 27: Pozivanje funkcije drveta odlučivanja sa ograničavanjem dubine (potkresivanjem) na 5 potomaka od korenog čvora

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 0.989
Rezultat test skupa: 0.973
Matrica kofuzije trening skupa:
[[531  10]
 [  1 477]]
Matrica kofuzije test skupa:
[[195   6]
 [  6 230]]
Vreme izvrsavanja: 1.781
```

Slika 18: Dtc metoda, originalni podaci, ginijev indeks, ograničena dubina

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 0.989
Rezultat test skupa: 0.963
Matrica kofuzije trening skupa:
[[533   8]
 [  3 475]]
Matrica kofuzije test skupa:
[[190  11]
 [  5 231]]
Vreme izvrsavanja: 1.907
```

Slika 19: Dtc metoda, originalni podaci, entropija, ograničena dubina

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.998
Matrica kofuzije trening skupa:
[[528   0]
 [  0 491]]
Matrica kofuzije test skupa:
[[214   0]
 [  1 222]]
Vreme izvrsavanja: 0.078
```

Slika 20: Dtc metoda, elementi van granice, ginijev indeks, ograničena dubina

```
-----Drvo odlucivanja:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.998
Matrica kofuzije trening skupa:
[[528   0]
 [  0 491]]
Matrica kofuzije test skupa:
[[214   0]
 [  1 222]]
Vreme izvrsavanja: 0.062
```

Slika 21: Dtc metoda, elementi van granice, entropija, ograničena dubina

3.1.3 Mašine sa potpornim vektorima

Metod binarne klasifikacije koji koristi linearni model $f_{w,w_0}(x) = w * x + w_0$. Dobijamo hiperravan koja deli podatke u dve klase. Neke podatke ne možemo razdvojiti pomoću hiperravni, tako da podatke moramo preslikati u visokodimenzioni prostor i onda ih razdvojiti pomoću hiperravni, koja će u dvodimenzionom prostoru biti odgovarajuća kriva. Funkcija koja radi preslikavanje instanci naziva se kernel.

```
from sklearn.svm import SVC
def svm(x_train, x_test, y_train, y_test, kernel, mode, rtype):
    print("-----SVM:-----")
    start = time.time()
    if mode == 'load':
        model = load(os.path.join('models', 'svm_' + rtype + '.joblib'))
    else:
        model = SVC(kernel=kernel, degree=2, gamma='scale')
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'svc_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 28: Definicija funkcije mašine sa potpornim vektorima

Konstruisanje modela klasifikacije mašina sa potpornim vektorima, pokušaćemo korišćenjem tri različita krenela: Gausovim('rbf'), linearnim('linear') i polinomijalnim('poly').

```
svm(x_train, x_test, y_train, y_test, kernel='rbf', mode='save', rtype='gen')
svm(x_train, x_test, y_train, y_test, kernel='linear', mode='save', rtype='gen')
svm(x_train, x_test, y_train, y_test, kernel='poly', mode='save', rtype='gen')
svm(ox_train, ox_test, oy_train, oy_test, kernel='rbf', mode='save', rtype='out')
svm(ox_train, ox_test, oy_train, oy_test, kernel='linear', mode='save', rtype='out')
svm(ox_train, ox_test, oy_train, oy_test, kernel='poly', mode='save', rtype='out')
```

Kod 29: Pozivanje funkcije mašine sa potpornim vektorima

```
-----SVM:-----
Rezultat trening skupa: 0.995
Rezultat test skupa: 0.989
Matrica kofuzije trening skupa:
[[507   5]
 [  0 507]]
Matrica kofuzije test skupa:
[[227   3]
 [  2 205]]
Vreme izvrsavanja: 42.924
```

Slika 22: Gausov kernel, originalni podaci

```
-----SVM:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[512   0]
 [  0 507]]
Matrica kofuzije test skupa:
[[229   1]
 [  1 206]]
Vreme izvrsavanja: 17.563
```

Slika 23: Linearan kernel, originalni podaci

```
-----SVM:-----
Rezultat trening skupa: 0.945
Rezultat test skupa: 0.931
Matrica kofuzije trening skupa:
[[456  56]
 [  0 507]]
Matrica kofuzije test skupa:
[[200  30]
 [  0 207]]
Vreme izvrsavanja: 34.904
```

Slika 24: Polinomijalni kernel, stepen 2, originalni podaci

```
-----SVM:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.792
Matrica kofuzije trening skupa:
[[531   0]
 [  0 488]]
Matrica kofuzije test skupa:
[[209   2]
 [ 89 137]]
Vreme izvrsavanja: 2.720
```

Slika 25: Gausov kernel, elementi van granice

```
-----SVM:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[531   0]
 [  0 488]]
Matrica kofuzije test skupa:
[[211   0]
 [  0 226]]
Vreme izvrsavanja: 0.167
```

Slika 26: Linearni kernel, elementi van granice

```
-----SVM:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[531   0]
 [  0 488]]
Matrica kofuzije test skupa:
[[209   2]
 [  0 226]]
Vreme izvrsavanja: 0.189
```

Slika 27: Polinomijalni kernel, stepen 2, elementi van granice

3.2 Ansambl klasifikacione tehnike

Ansambl tehnike se nazivaju i meta klasifikacione metode, zato što ne možemo odmah napraviti model klasifikacije, već moramo konstruisati nekoliko jednostavnih modela, pomenutih u odeljku [3.1] čije rezultate ove tehnike kombinuju u cilju smanjenja nivoa greške.

3.2.1 Nasumična šuma

Deli skup podataka na komplementarne podskupove i za svaki od podskupova, generiše zaseban model drveta odlučivanja. Krajnji model predstavlja srednju vrednost rezultata dobijenih iz generisanih modela. Kako smo pojedinačno drvo odlučivanja već obradili u odeljku [3.1.2] iz dobijenih rezultata možemo zaključiti da metod najbolje radi korišćenjem Ginijevog indeksa, uz ograničavanje maksimalne dubine stabla, jer u suprotnom može doći do preprilagođavanja podacima.

```
from sklearn.ensemble import RandomForestClassifier
def rfc(x_train, x_test, y_train, y_test, n_est, mode, rtype):
    print("-----Nasumicna suma:-----")
    start = time.time()
    if mode == 'load':
        model = load(os.path.join('models', 'rfc_' + rtype + '.joblib'))
    else:
        model = RandomForestClassifier(n_estimators=n_est, max_depth=5, criterion='gini')
    # noinspection PyTypeChecker
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'rfc_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 30: Definicija funkcije nasumična šuma

Jedan novi parametar koji trebamo poslati funkcija je broj drveta odlučivanja koje treba napraviti. Konstruisaćemo i uporediti rezultate modela za 10, 50 i 100 drveta odlučivanja.

```
rfc(x_train, x_test, y_train, y_test, n_est=10, mode='save', rtype='gen')
rfc(x_train, x_test, y_train, y_test, n_est=50, mode='save', rtype='gen')
rfc(x_train, x_test, y_train, y_test, n_est=100, mode='save', rtype='gen')
rfc(ox_train, ox_test, oy_train, oy_test, n_est=10, mode='save', rtype='out')
rfc(ox_train, ox_test, oy_train, oy_test, n_est=50, mode='save', rtype='out')
rfc(ox_train, ox_test, oy_train, oy_test, n_est=100, mode='save', rtype='out')
```

Kod 31: Pozivanje funkcije nasumična šuma

```
-----Nasumicna suma:-----
Rezultat trening skupa: 0.986
Rezultat test skupa: 0.963
Matrica kofuzije trening skupa:
[[503  14]
 [  0 502]]
Matrica kofuzije test skupa:
[[210  15]
 [  1 211]]
Vreme izvrsavanja: 0.913
```

Slika 28: Rfc, 10 drveta, originalni podaci

```
-----Nasumicna suma:-----
Rezultat trening skupa: 0.991
Rezultat test skupa: 0.968
Matrica kofuzije trening skupa:
[[508   9]
 [  0 502]]
Matrica kofuzije test skupa:
[[214  11]
 [  3 209]]
Vreme izvrsavanja: 0.770
```

Slika 29: Rfc, 50 drveta, originalni podaci

```
-----Nasumicna suma:-----
Rezultat trening skupa: 0.993
Rezultat test skupa: 0.977
Matrica kofuzije trening skupa:
[[510   7]
 [  0 502]]
Matrica kofuzije test skupa:
[[215  10]
 [  0 212]]
Vreme izvrsavanja: 1.284
```

Slika 30: Rfc, 100 drveta, originalni podaci

```
-----Nasumicna suma:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[514   0]
 [  0 505]]
Matrica kofuzije test skupa:
[[228   0]
 [  0 209]]
Vreme izvrsavanja: 0.106
```

Slika 31: Rfc, 10 drveta, elemnti van granica

```
-----Nasumicna suma:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[514   0]
 [  0 505]]
Matrica kofuzije test skupa:
[[228   0]
 [  0 209]]
Vreme izvrsavanja: 0.235
```

Slika 32: Rfc, 50 drveta, elemnti van granica

```
-----Nasumicna suma:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[514   0]
 [  0 505]]
Matrica kofuzije test skupa:
[[228   0]
 [  0 209]]
Vreme izvrsavanja: 0.390
```

Slika 33: Rfc, 100 drveta, elemnti van granica

3.2.2 Pakovanje

Ansambl tehnika koja deli ulazni skup podataka na podskupove u kojima se elementi mogu ponavljati i za svaki skup formira zaseban model. Krajnji model se formira računanjem srednje vrednosti svih prethodno formiranih parcijalnih modela. Testiraćemo ovu tehniku korišćenjem 2 osnovna modela. Jedan će biti drvo odlučivanja sa ograničenom dubinom, a drugi mašina sa potpornim vektorima koja koristi linearni kernel.

```
from sklearn.ensemble import BaggingClassifier
def bag(x_train, x_test, y_train, y_test, n_est, mode, rtype, model):
    print("-----Bagging:-----")
    start = time.time()
    if mode == 'load':
        model = load(os.path.join('models', 'bagging_' + rtype + '.joblib'))
    else:
        model = BaggingClassifier(base_estimator=model, n_estimators=n_est)
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'bagging_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 32: Definicija funkcije pakovanja

Pakovanje sa drvetom odlučivanja kao primarnom metodom ćemo pozivati sa 10 i 50 različitih modela, dok ćemo kod mašina sa potpornim vektorima koristiti 5 i 20 modela.

```
bag(x_train, x_test, y_train, y_test, n_est=10, mode='save', rtype='gen',
    model = DecisionTreeClassifier(max_depth=5))
bag(x_train, x_test, y_train, y_test, n_est=50, mode='save', rtype='gen',
    model = DecisionTreeClassifier(max_depth=5))
bag(x_train, x_test, y_train, y_test, n_est=5, mode='save', rtype='gen',
    model = SVC(kernel='linear', gamma='scale'))
bag(x_train, x_test, y_train, y_test, n_est=20, mode='save', rtype='gen',
    model = SVC(kernel='linear', gamma='scale'))
bag(ox_train, ox_test, oy_train, oy_test, n_est=10, mode='save', rtype='out',
    model = DecisionTreeClassifier(max_depth=5))
bag(ox_train, ox_test, oy_train, oy_test, n_est=50, mode='save', rtype='out',
    model = DecisionTreeClassifier(max_depth=5))
bag(ox_train, ox_test, oy_train, oy_test, n_est=5, mode='save', rtype='out',
    model = SVC(kernel='linear', gamma='scale'))
bag(ox_train, ox_test, oy_train, oy_test, n_est=20, mode='save', rtype='out',
    model = SVC(kernel='linear', gamma='scale'))
```

Kod 33: Pozivanje funkcije pakovanja

```
-----Bagging:-----
Rezultat trening skupa: 0.994
Rezultat test skupa: 0.986
Matrica kofuzije trening skupa:
[[513   2]
 [  4 500]]
Matrica kofuzije test skupa:
[[226   1]
 [  5 205]]
Vreme izvrsavanja: 16.379
```

Slika 34: Pakovanje, 10 drveta odlučivanja, originalni podaci

```
-----Bagging:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.991
Matrica kofuzije trening skupa:
[[515   0]
 [  0 504]]
Matrica kofuzije test skupa:
[[225   2]
 [  2 208]]
Vreme izvrsavanja: 83.830
```

Slika 35: Pakovanje, 50 drveta odlučivanja, originalni podaci

```
-----Bagging:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.993
Matrica kofuzije trening skupa:
[[515   0]
 [  0 504]]
Matrica kofuzije test skupa:
[[227   0]
 [  3 207]]
Vreme izvrsavanja: 85.470
```

Slika 36: Pakovanje, 5 mašina sa potpornim vektorima, originalni podaci

```
-----Bagging:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[515   0]
 [  0 504]]
Matrica kofuzije test skupa:
[[227   0]
 [  2 208]]
Vreme izvrsavanja: 325.483
```

Slika 37: Pakovanje, 20 mašina sa potpornim vektorima, originalni podaci

```
-----Bagging:-----
Rezultat trening skupa: 0.999
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[513   0]
 [  1 505]]
Matrica kofuzije test skupa:
[[229   0]
 [  2 206]]
Vreme izvrsavanja: 0.658
```

Slika 38: Pakovanje, 10 drveta odlučivanja, elementi van granica

```
-----Bagging:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[513   0]
 [  0 506]]
Matrica kofuzije test skupa:
[[229   0]
 [  2 206]]
Vreme izvrsavanja: 3.271
```

Slika 39: Pakovanje, 50 drveta odlučivanja, elementi van granica

```
-----Bagging:-----
Rezultat trening skupa: 0.999
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[513   0]
 [  1 505]]
Matrica kofuzije test skupa:
[[229   0]
 [  0 208]]
Vreme izvrsavanja: 0.908
```

Slika 40: Pakovanje, 5 mašina sa potpornim vektorima, elementi van granica

```
-----Bagging:-----
Rezultat trening skupa: 0.999
Rezultat test skupa: 0.998
Matrica kofuzije trening skupa:
[[513   0]
 [  1 505]]
Matrica kofuzije test skupa:
[[228   1]
 [  0 208]]
Vreme izvrsavanja: 3.879
```

Slika 41: Pakovanje, 20 mašina sa potpornim vektorima, elementi van granica

3.2.3 Pojačavanje

Na početku se formira loš klasifikator i svim slogovima se dodaju jednake težine. Kroz iteracije se vrši prepravka težina na osnovu rezultata iz prethodne iteracije. Ako je podatak tačno klasifikovan, težina sloga se smanjuje, dok ako je klasifikovan pogrešno, težina se povećava. Glavna ideja iz ovog meta-klasifikatora je da na osnovu nekoliko slabih klasifikatora, napravi jedan jak. Kako tehnika pojačavanja ne može da radi sa mašinama sa potpornim vektorima, kao osnovni model ćemo koristiti samo drvo odlučivanja sa ograničenom dubinom.

```
from sklearn.ensemble import AdaBoostClassifier
def boost(x_train, x_test, y_train, y_test, n_est, mode, rtype):
    print("-----Boosting:-----")
    start = time.time()
    model = DecisionTreeClassifier(max_depth=5)
    if mode == 'load':
        model = load(os.path.join('models', 'boosting_' + rtype + '.joblib'))
    else:
        model = AdaBoostClassifier(base_estimator=model, n_estimators=n_est)
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'boosting_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 34: Definisanje funkcije pojačavanja

Navedenu funkciju pozvaćemo sa 10, 50 i 100 različitih modela drveta odlučivanja.

```
boost(x_train, x_test, y_train, y_test, n_est=10, mode='save', rtype='gen')
boost(x_train, x_test, y_train, y_test, n_est=50, mode='save', rtype='gen')
boost(x_train, x_test, y_train, y_test, n_est=100, mode='save', rtype='gen')
boost(ox_train, ox_test, oy_train, oy_test, n_est=10, mode='save', rtype='out')
boost(ox_train, ox_test, oy_train, oy_test, n_est=50, mode='save', rtype='out')
boost(ox_train, ox_test, oy_train, oy_test, n_est=100, mode='save', rtype='out')
```

Kod 35: Pozivanje funkcije pojačavanja

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.986
Matrica kofuzije trening skupa:
[[534   0]
 [  0 485]]
Matrica kofuzije test skupa:
[[206   2]
 [  4 225]]
Vreme izvrsavanja: 18.188
```

Slika 42: Pojačavanje, 10 drveta odlučivanja, originalni podaci

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[534   0]
 [  0 485]]
Matrica kofuzije test skupa:
[[207   1]
 [  1 228]]
Vreme izvrsavanja: 89.321
```

Slika 43: Pojačavanje, 50 drveta odlučivanja, originalni podaci

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[534   0]
 [  0 485]]
Matrica kofuzije test skupa:
[[208   0]
 [  0 229]]
Vreme izvrsavanja: 187.501
```

Slika 44: Pojačavanje, 100 drveta odlučivanja, originalni podaci

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[507   0]
 [  0 512]]
Matrica kofuzije test skupa:
[[235   0]
 [  2 200]]
Vreme izvrsavanja: 0.141
```

Slika 45: Pojačavanje, 10 drveta odlučivanja, elementi van granice

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[507   0]
 [  0 512]]
Matrica kofuzije test skupa:
[[235   0]
 [  2 200]]
Vreme izvrsavanja: 0.110
```

Slika 46: Pojačavanje, 50 drveta odlučivanja, elementi van granice

```
-----Boosting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 0.995
Matrica kofuzije trening skupa:
[[507   0]
 [  0 512]]
Matrica kofuzije test skupa:
[[235   0]
 [  2 200]]
Vreme izvrsavanja: 0.176
```

Slika 47: Pojačavanje, 100 drveta odlučivanja, elementi van granice

3.2.4 Glasanje

Svaki ulazni slog se klasifikuje svim prosleđenim osnovnim modelima i na osnovu dobijenih rezultata određuje se klasa svakog sloga. U našem primeru koristićemo tri modela: nasumičnu šumu koju formiramo pomoću 100 drveta odlučivanja sa ograničenom dubinom, mašinu sa potpornim vektorima uz korišćenje linearnog kernela i drvo odlučivanja sa ograničenom dubinom.

```
from sklearn.ensemble import VotingClassifier
def vot(x_train, x_test, y_train, y_test, power, mode, rtype):
    print("-----Voting:-----")
    start = time.time()
    est_model1 = RandomForestClassifier(n_estimators=100, max_depth=5)
    est_model2 = SVC(kernel='linear', gamma='scale')
    est_model3 = DecisionTreeClassifier(max_depth=5)
    if mode == 'load':
        model = load(os.path.join('models', 'voting_' + rtype + '.joblib'))
    else:
        model = VotingClassifier(estimators=[('dtc', est_model1), ('svc', est_model2),
                                             ('tree', est_model3)],
                                 voting=power)
    est_model1.fit(x_train, y_train.ravel())
    est_model2.fit(x_train, y_train.ravel())
    est_model3.fit(x_train, y_train.ravel())
    fit_model(model, x_train, x_test, y_train, y_test, mode, 'voting_' + rtype)
    prediction(model, x_train, x_test, y_train, y_test)
    print(f"Vreme izvrsavanja: {time.time()-start:.3f} \n")
```

Kod 36: Definicija funkcije galsanja

Glasanje u oba poziva funkcija je “hard”, što znači da će rezultati tri osnovna modela biti upoređivani na svakom slogu i ona klasa koja ima više glasova, da se podsetimo postoje dve klase, biće dodeljena posmatranom slogu.

```
vot(x_train, x_test, y_train, y_test, power='hard', mode='save', rtype='gen')
vot(ox_train, ox_test, oy_train, oy_test, power='hard', mode='save', rtype='out')
```

Kod 37: Pozivanje funkcije glasanja

```
-----Voting:-----
Rezultat trening skupa: 0.998
Rezultat test skupa: 0.998
Matrica kofuzije trening skupa:
[[508   2]
 [  0 509]]
Matrica kofuzije test skupa:
[[231   1]
 [  0 205]]
Vreme izvrsavanja: 24.229
```

Slika 48: Glasanje, originalni podaci

```
-----Voting:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[516   0]
 [  0 503]]
Matrica kofuzije test skupa:
[[226   0]
 [  0 211]]
Vreme izvrsavanja: 1.031
```

Slika 49: Glasanje, elementi van granica

4 Analiza i poređenje dobijenih rezultata

Nakon prikazanih rezultata izvršavanja jednostavnih modela i modela dobijenih pomoću ansambl tehnika, sada ćemo sve dobijene rezultate posebno analizirati i upoređivati na osnovu različitih parametara koji su korišćeni u konstruisanju tih modela kako bismo videli koji modeli i sa kojim parametrima su dali najbolje rezultate i koliko vremena je bilo potrebno za njihovo generisanje. Na kraju ćemo uporediti najbolje rezultate jednostavnih metoda [4.8], kao i najbolje rezultate ansambl tehnika [4.9]. U tabelama će najbolji rezultati biti obojeni plavom bojom.

4.1 K najbližih suseda

Broj suseda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
3	0.979	0.977	107.122
5	0.968	0.974	105.178
10	0.965	0.967	106.348

Tabela 1: Rezultati metoda k najbližih suseda nad originalnim podacima

Broj suseda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
3	0.984	0.941	1.707
5	0.975	0.920	2.064
10	0.970	0.927	2.227

Tabela 2: Rezultati metoda k najbližih suseda nad elementima van granice

Možemo primetiti da se povećanjem broja suseda, rezultati klasifikovanja pogoršavaju. Razlog slabljenja rezultata je nepostojanje zajedničke karakteristike većeg broja slogova u n-dimenzionom prostoru, koja bi odvojila dve klase na precizniji način. Iz rezultata možemo zaključiti da je od ispitanih modela najbolji sa 3 suseda.

4.2 Drvo odlučivanja

Mera nečistoće	Dubina drveta	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Gini	Neograničena	1.000	0.970	1.787
Entropija	Neograničena	1.000	0.970	1.851
Gini	5	0.989	0.973	1.781
Entropija	5	0.989	0.963	1.907

Tabela 3: Rezultati metoda drvo odlučivanja nad originalnim podacima

Mera nečistoće	Dubina drveta	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Gini	Neograničena	1.000	0.995	0.063
Entropija	Neograničena	1.000	0.998	0.109
Gini	5	1.000	0.998	0.078
Entropija	5	1.000	0.998	0.062

Tabela 4: Rezultati metoda drvo odlučivanja nad elementima van granice

Kao što smo naveli u odeljku [3.1.2] konstruisanje modela nad podacima bez ograničavanja dubine dovodi do blagog preprilagođavanja podacima, što je uočljivije kod originalnih podataka nego kod elemenata van granice. Uz ograničavanje dubine (odsecanje stabla) kod testiranja nad originalnim podacima dobili smo za nijansu lošije rezultate na trening skupu, ali istovremeno i bolje rezultate na test skupu uz korišćenje Ginijevog indeksa kao meru nečistoće. Bolji rezultati i kraće vreme izvršavanja uz korišćenje ograničenja dubine možemo primetiti i kod testiranja modela nad elementima van granice, čime se upotpunjuje ranije tvrdnja da se dobijaju bolji modeli korišćenjem odsecanja stabla.

4.3 Mašine sa potpornim vektorima

Kernel	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Gausov	0.995	0.989	42.924
Polinomijalan	0.945	0.931	34.904
Linearan	1.000	0.995	17.563

Tabela 5: Rezultati metoda mašine sa potpornim vektorima nad originalnim podacima

Kernel	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Gausov	1.000	0.792	2.720
Polinomijalan	1.000	0.995	0.189
Linearan	1.000	1.000	0.167

Tabela 6: Rezultati metoda mašine sa potpornim vektorima nad elementima van granice

Kod testiranja nad originalnim podacima, ne možemo mnogo toga zaključiti na osnovu rezultata, osim da je pomoću linearanog kernela generisan skoro nepogrešiv model i da mu je bilo potrebno duplo manje vremena u odnosu na polinomijalan kernel i još više u odnosu na Gausov, bolje zaključke možemo izvući ako budemo posmatrali rezultate generisanja modela nad elementima van granice. Pre svega kod Gausovog kernela možemo primetiti očigledno preprilagođavanje podacima, što i nije iznenađujuće s obzirom da na takav način funkcioniše ovaj algoritam, a uz to testirali smo nad elementima van granice koji se dosta razlikuju jedni od drugih, pa je ovakav rezultat i očekivan. Pored toga vidimo da je i dosta više vremena utrošeno na generisanje modela mašine sa potpornim vektorima uz korišćenje Gausovog kernela zbog prilagođavanja podacima. I u ovom slučaju možemo ustanoviti da je linearni kernel dao najbolje, čak i nepogrešive, rezultate za najkraće vreme u poređenju sa drugim kernelima.

4.4 Nasumična šuma

Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
10	0.986	0.963	0.913
50	0.991	0.968	0.770
100	0.993	0.977	1.284

Tabela 7: Rezultati metoda nasumična šuma nad originalnim podacima

Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
10	1.000	1.000	0.106
50	1.000	1.000	0.235
100	1.000	1.000	0.390

Tabela 8: Rezultati metoda nasumične šume nad elementima van granice

Upoređivanjem rezultata nasumične šume i jednog drveta odlučivanja nad originalnim podacima, možemo zaključiti da smo tek upotrebom sto različitih stabala dobili bolje rezultate uz kraće vreme izvršavanja. Što se elemenata van granice tiče, generisanje modela je trajalo duže u odnosu na jedno drvo odlučivanja, ali je uvek davalo besprekorne rezultate.

4.5 Pakovanje

Osnovni model	Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Drvo odlučivanja	10	0.994	0.986	16.379
Drvo odlučivanja	50	1.000	0.991	83.830
SVC	5	1.000	0.993	85.470
SVC	20	1.000	0.995	325.483

Tabela 9: Rezultati metoda pakovanja nad originalnim podacima

Osnovni model	Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
Drvo odlučivanja	10	0.999	0.995	0.658
Drvo odlučivanja	50	1.000	0.995	3.271
SVC	5	0.999	1.000	0.908
SVC	20	0.999	0.998	3.879

Tabela 10: Rezultati metoda pakovanja nad elementima van granice

Prvu stvar koju možemo primetiti kod modela je proporcionalno vreme izvšavanja broju osnovnih modela korišćenih za generisanje. Modeli konstruisani pomoću mašina sa potponrim vektorima su dali bolje rezultate u odnosu na one sa drvećem odlučivanja, uz duže vreme izvršavanja. Zanimljivu stvar možemo primetit kod testiranja modela konsturisanog pomoću pet mašina sa potpornim vektorima na elementima van granice, gde je model pogrešio prilikom klasifikovanja jednog sloga nad trening podacima, a test podatke je klasifikovao bez greške.

4.6 Pojačavanje

Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
10	1.000	0.986	18.188
50	1.000	0.995	89.321
100	1.000	1.000	187.501

Tabela 11: Rezultati metoda pojačavanja nad originalnim podacima

Broj modela	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
10	1.000	0.995	0.141
50	1.000	0.995	0.110
100	1.000	0.995	0.176

Tabela 12: Rezultati metoda pojačavanja nad elementima van granice

Ansambl tehnika pojačavanja je takođe dala veoma precizne rezultate. Možemo videti povećanje vremena izvršavanja uz porast broja osnovnih modela. Zanimljivu stvar primećujemo kod rezultata generisanja modela nad elementima van granice, a to je da su rezultati sva tri modela identična, jedina stvar po kojoj se razlikuje je vreme izvršavanja, ali i ta razlika je minimalna.

4.7 Glasanje

Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
0.998	0.998	24.229

Tabela 13: Rezultat izvršavanja metoda glasanja nad originalnim podacima

Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja
1.000	1.000	1.031

Tabela 14: Rezultat izvršavanja metoda glasanja nad elementima van granice

Ne možemo previše komentarisati dobijene rezultate, osim da kažemo da su izuzetno dobri. Upoređivanje metode glasanja sa ostalim ansambl tehnikama ćemo uraditi u odeljku [4.9].

4.8 Jednostavne metode

Nakon što smo završili analizu svih metoda pojedinačno, sada ćemo analizirati grupe metoda, da bismo videli koje su se najbolje pokazale nad našim podacima. Za svaku metodu uzimamo parametre sa kojima se najbolje pokazala u prethodnom odeljku, za obe grupe podataka posebno.

Što se tiče originalnih podataka, za k najbližih suseda uzimamo tri najbliža suseda, kod drveta odlučivanja uzimamo Ginijev indeks kao meru nečistoće i ograničavamo drvo do petog potomka, dok kod mašina sa potpornim vektorima koristimo linearni kernel.

Kod poređenja rezultata nad elementima van granice, uzimamo 3 najbliža suseda, drvo odlučivanja sa ograničenjem do petog nivoa sa entropijom kao merom nečistoće i mašinu sa potpornim vektorima uz korišćenje linearnog kernela.

Metoda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja	Tačno klasifikovani	Pogrešno klasifikovani
Knn	0.979	0.977	107.122	998 + 427	21 + 10
Drvo odlučivanja	0.989	0.973	1.781	1008 + 425	11 + 12
Svm	1.000	0.995	17.563	1019 + 435	0 + 2

Tabela 15: Rezultati jednostavnih metoda nad originalnim podacima

Metoda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja	Tačno klasifikovani	Pogrešno klasifikovani
Knn	0.984	0.941	1.707	1003 + 411	16 + 26
Drvo odlučivanja	1.000	0.998	0.062	1019 + 436	0 + 1
Svm	1.000	1.000	0.167	1019 + 437	0 + 0

Tabela 16: Rezultati jednostavnih metoda nad elementima van granice

U tabelama su, kao i do sada, posebno označeni metodi koji su dali sveukupno najbolje rezultate – plavom bojom i oni koji su dali najbolje rezultate u odnosu na vreme generisanja – zelenom bojom. Ukoliko u tabeli ne postoji red obojen zelenom bojom, znači da je metod sa najboljim rezultatom ujedno i metod sa najboljim odnosom rezultata i vremena izvršavanja.

Kod rezultata svm metoda vidimo veoma malu razliku između trening i test skupa, tako da možemo tvrditi da je generisan odličan model i da nismo pogrešili izborom linearnog jezgra. Jedina mana ovog metoda je nešto duže vreme generisanja modela u odnosu na drvo odlučivanja kod originalnih podataka.

4.9 Ansambl tehnike

Kao i u delu [4.8] i ovde upoređujemo rezultate pojedinačnih modela, sada iz ansambl grupe, uzimamo samo po jedan model sa parametrima koji su dali najbolje rezultate. Za originalne podatke, nasumičnu šumu generišemo sa 50 stabala bez dubinskog ograničenja, metod pakovanja izvršavamo pomoću 20 drveta odlučivanja, dok kod pojačavanja to radimo sa 50. Na kraju generisanje modela tehnike glasanja postižemo pomoću 3 različita modela, nasumična šuma (100 stabala), drvo odlučivanja sa ograničenom dubinom i mašine sa potpornim vektorima (linearni kernel).

Metoda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja	Tačno klasifikovani	Pogrešno klasifikovani
Nasumična šuma	0.993	0.977	1.284	1012 + 427	7 + 10
Pakovanje	1.000	0.995	325.483	1019 + 435	0 + 2
Pojačavanje	1.000	1.000	187.501	1019 + 437	0 + 0
Glasanje	0.998	0.998	24.229	1017 + 436	2 + 1

Tabela 17: Rezultati ansambl metoda nad originalnim podacima

Metoda	Rezultat trening skupa	Rezultat test skupa	Vreme izvršavanja	Tačno klasifikovani	Pogrešno klasifikovani
Nasumična šuma	1.000	1.000	0.106	1019 + 437	0 + 0
Pakovanje	0.999	1.000	0.908	1018 + 437	1 + 0
Pojačavanje	1.000	0.995	0.110	1019 + 435	0 + 2
Glasanje	1.000	1.000	1.031	1019 + 437	0 + 0

Tabela 18: Rezultati ansambl metoda nad elementima van granica

Kao što je i očekivano, kod ansambl tehnika dobili smo bolje rezultate u poređenju sa jednostavnim metodama, razlog tome je generisanje većeg broja jednostavnih modela radi dobijanja preciznijih podataka, kako na trening tako i na test skupu.

Nad originalnim podacima najbolje rezultate je dala tehnika pojačavanja koja je klasifikovala podatke bez greške, ali je trebalo dosta vremena za generisanje modela. Nasumična šuma je dala bolje rezultate od drveta odlučivanja u kraćem vremenskom roku.

Kod klasifikacije elemenata van granice možemo videti da su sve tehnike konstruisale skoro nepogrešive modele u vremenu oko i ispod jedne sekunde.

5 Zaključak

Kada pogledamo sveobuhvatne rezultate prvo što možemo zaključiti je da je metod k najbližih suseda dao najlošije rezultate nad oba skupa podataka. Ako posmatramo samo elemente van granica preostali metodi su dali skoro nepogrešive rezultate sa sličnim vremenom potrebnim za konstruisanje modela, tako da ćemo više pažnje posvetiti rezultatima nad originalnim podacima.

Drvo odlučivanja i nasumična šuma su dali dobre rezultate za veoma kratko vreme u poređenju sa vremenima konstruisanja drugih modela, ali isto tako ako uporedimo njihove rezultate sa rezultatima ostalih modela možemo zaključiti da su dosta lošiji klasifikatori pogotovo posmatrajući test skupove. Preostala četiri modela (svm, pakovanje, pojačavanje, glasanje) su dali slične, skoro nepogrešive rezultate. Najbolji od njih je model konstruisan tehnikom pojačavanja koji je perfektno klasifikovao podatke, ali uz nekoliko puta duže vreme konstruisanja u odnosu na mašine sa potpornim vektorima i glasanjem. Ukoliko posmatramo rezultate modela uz vreme potrebno za njihovo konstruisanje, možemo zaključiti da je najbolje podatke dala metoda mašina sa potpornim vektorima uz korišćenje linearnog kernela, koja je tačno klasifikovala sve podatke na trening skupu, dok je na test skupu pogrešno klasifikovala samo dva podatka od 437 koliko je ukupno bilo u tom skupu.

Još jedna zanimljiva stvar koju možemo primetiti je efikasnost i preciznost klasifikovanja elemenata van granice. Iz ovoga možemo zaključiti da iako su vrednosti podataka u njima odudarali od vrednosti originalnih podataka i dalje smo precizno mogli da klasifikujemo podatke u jednu od dve klase. Pokušaćemo da klasifikujemo podatke zajedno sa elementima van granice, da bismo potvrdili ovu pretpostavku. Klasifikovaćemo elemente iz [Kod 10] pomoću metode mašina sa potpornim vektorima korišćenjem linearnog kerenela [Kod 28] jer je ona pokazala najbolje rezultate do sada.

```
-----SVM:-----
Rezultat trening skupa: 1.000
Rezultat test skupa: 1.000
Matrica kofuzije trening skupa:
[[534   0]
 [  0 485]]
Matrica kofuzije test skupa:
[[208   0]
 [  0 229]]
Vreme izvrsavanja: 6.584
```

Slika 50: Rezultat izvršavanja mašine sa potpornim vektorima uz korišćenje linearnog kernela nad originalnim podacima sa elementim van granice

Vidimo da je metod klasifikovao podatke bez ijedne greške, ali ono što je još zanimljivije je vreme generisanja modela, jer kao što vidimo na [Slika 23] vreme potrebno za generisanje modela je bilo 17.563s dok je ovde 6.584s, skoro 3 puta manje iz čega možemo zaključiti da model radi najbolje nad podacima bez isključivanja elementima van granice.

6 Literatura

Aggarwal, C. C. (2014). *Data Classification: Algorithms and Applications.* Chapman and Hall/CRC.

Kantardzic, M. (2011). *Data Mining: Concepts, Models, Methods, and Algorithms, Second Edition: Concepts, Models, Methods, and Algorithms.* John Wiley & Sons.

Pedregosa, V. G. (2011). *scikit-learn*. Retrieved from scikit-learn: https://scikit-learn.org/stable/index.html

Eric Jones, T. O. (2001). *SciPy*. Retrieved from SciPy: https://scipy.org/