som-projekat

June 15, 2023

1 Projektni zadatak iz predmeta Sistemi odlučivanja u medicini

Nataša Jovanović, 2020/0100 Marija Nedeljković, 2020/0096

[ ]:
```
# učitavanje biblioteka
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

from sklearn.model_selection import KFold
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.tree import DecisionTreeClassifier
```

2 1. Učitavanje i pretprocesiranje podataka

[ ]:
```
data = pd.read_csv('03_bone_marrow_dataset.csv')
print("Dimenzije skupa podataka: ", data.shape)
```

```
Dimenzije skupa podataka:  (187, 37)
```

[ ]:
```
# brišemo razmak na kraju naziva atributa (lakše je za kasniju analizu)
for attribute in data.columns:
    if(attribute[-1] == ' '):
        data.rename(columns = {attribute:attribute[0:-1]}, inplace = True)
attributes = data.columns
data
```

[ ]:

	Recipientgender	Stemcellsource	Donorage numeric	Donorage35	IIIV	\
0	1	1	22.830137	0	1	
1	1	0	23.342466	0	1	

```
2                  1               0         26.394521           0     1
3                  0               0         39.684932           1     1
4                  0               1         33.358904           0     0
..               …               …             …         …   …
182                1               1         37.575342           1     1
183                0               1         22.895890           0     0
184                0               1         27.347945           0     1
185                1               1         27.780822           0     1
186                1               1         55.553425           1     1

     Gendermatch  DonorABO RecipientABO RecipientRh ABOmatch  … extcGvHD  \
0              0         1            1           1        0  …        1
1              0        -1           -1           1        0  …        1
2              0        -1           -1           1        0  …        1
3              0         1            2           1        1  …        ?
4              0         1            2           0        1  …        1
..           …       …          …         …       …  …      …
182            0         1            1           0        0  …        1
183            0         1            0           1        1  …        1
184            0         1           -1           1        1  …        1
185            0         1            0           1        1  …        0
186            0         1            2           1        1  …        1

    CD34kgx10d6 numeric CD3dCD34 numeric CD3dkgx10d8 numeric  \
0                  7.20          1.33876                5.38
1                  4.50         11.078295               0.41
2                  7.94         19.01323                0.42
3                  4.25         29.481647               0.14
4                 51.85          3.972255              13.05
..                  …               …                  …
182               11.08          2.52275                4.39
183                4.64          1.038858               4.47
184                7.73          1.635559               4.73
185               15.41          8.07777                1.91
186                9.91          0.948135              10.45

     Rbodymass numeric  ANCrecovery numeric  PLTrecovery numeric  \
0                   35                   19                   51
1                 20.6                   16                   37
2                 23.4                   23                   20
3                   50                   23                   29
4                    9                   14                   14
..                 …                   …                  …
182                 44                   15                   22
183               44.5                   12                   30
184                 33                   16                   16
185                 24                   13                   14
```

```
186                 37                  18                  20

     time_to_aGvHD_III_IV numeric  survival_time numeric  \
0                                32                   999
1                           1000000                   163
2                           1000000                   435
3                                19                    53
4                           1000000                  2043
..                              …                     …
182                              16                   385
183                         1000000                   634
184                         1000000                  1895
185                              54                   382
186                         1000000                  1109

    survival_status numeric
0                           0
1                           1
2                           1
3                           1
4                           0
..                        …
182                         1
183                         1
184                         0
185                         1
186                         0

[187 rows x 37 columns]
```

Podaci koji se nalaze u bazi:

1. Recipientgender - Pol pacijenta: 0 ili 1

2. Stemcellsource - Izvor ćelija (periferna krv, koštana srž): 0 ili 1
3. Donorage numeric - Starost davaoca u vreme afereze hematopoetskih matičnih ćelija: float num 18.646575 - 55.553425
4. Donorage35 - Da li je starost donora manja od 35 godina: 0 ili 1
5. IIIV - razvoj bolesti transplantata 3. ili 4. stepena: 0 ili 1
6. Gendermatch - kompatibilnost donora i primaoca prema polu: 0 ili 1
7. DonorABO - krvna grupa donora: -1, 0, 1, 2
8. RecipientABO - krvna grupa pacijenta: -1, 0, 1, 2
9. RecipientRh - da li pacijent ima Rh raktor: 0 ili 1
10. ABOmatch - kompatibilnost pacijenta i donora po krvnoj grupi: 0 ili 1
11. CMVstatus - kompatibilnost pacijenta i donora prema citomegalo virusu koji se može razviti (što je veća kompatibilnost to je lošije po pacijenta): num 0, 1, 2, 3 i ?
12. DonorCMV - postojanje citomegalo virusne infekcije kod donora pre transplantacije: 0 ili 1
13. RecipientCMV - postojanje citomegalo virusne infekcije kod pacijenta pre transplantacije: 0

ili 1

14. Disease - vrsta bolesti: ALL, AML, chronic, nonmalignant, lymphoma
15. Riskgroup - rizična grupa: 0 ili 1
16. Txpostrelapse - da li je rađena druga transplantacija nakon što je prva prošla loše: 0 ili 1
17. Diseasegroup - da li je oboljenje maligno ili nije: 0 ili 1
18. HLAmatch - grupe antigena donora i primaoca: 0, 1, 2, 3
19. HLAmismatch - da li su se antigeni poklopili: 0 ili 1
20. Antigen - u koliko antigena postoji razlika između donora i pacijenta: -1, 0, 1, 2 i ?
21. Alel - u koliko allela postoji razlika između donora i pacijenta: -1, 0, 1, 2, 3, ?
22. HLAgrI - razlika između donora i primaoca u parametrima kao što su antigen, alel, HLA, itd: 0, 1, 2, 3, 4, 5, 6, 7
23. Recipientage numeric - starost pacijenta u trenutku transplantacije: 0.6 - 20.2
24. Recipientage10 - da li je starost pacijenta manja od 10 godina: 0 ili 1
25. Recipientageint - godište pacijenta diskretizovano po intervalima: 0, 1, 2
26. Relapse - povratak bolesti: 0 ili 1
27. aGvHDIIIIV - da li se razvila neka bolest 3. i 4. stepena: 0 ili 1
28. extcGvHD - da li se razvila neka bolest 3. i 4. stepena: 0, 1 i ?
29. CD34kgx10d6 numeric - doza neke vrste ćelija skalirana po težini čoveka: float 0.79 - 57.78
30. CD3dCD34 numeric - doza druge vrste ćelija skalirana po dozi druge vrste: float 0 - 99.56097 i ?
31. CD3dkgx10d8 numeric - doza neke vrste ćelija skalirana po težini čoveka: float 0 - 20.02 i ?
32. Rbodymass numeric - bodymass indeks pacijenta: float 0 - 103.4 i ?
33. ANCrecovery numeric - oporavak neutrofila definisan kao njihov broj (>0.5 je da) : float 9 - 26.0 i 1000000.0
34. PLTrecovery numeric - oporavak trombocita definisam njihov broj (>50000 da): float 9 - 285 i 1000000.0
35. time_to_aGvHD_III_IV numeric - vreme u danima proteklo do razvoja akutne bolesti transplantata 3. ili 4. stepena: 10 - 100 i 1000000
36. survival_time numeric - vreme posmatranja (u slučaju živog pacijenta) ili vreme do smrti: 6 - 3364
37. survival_status numeric - da li je osoba živa ili mrtva: 0 ili 1

[ ]:
```
print(set(data.dtypes))
data.dtypes[data.dtypes == 'O']
```

```
{dtype('float64'), dtype('O'), dtype('int64')}
```

[ ]:
```
RecipientABO          object
RecipientRh           object
ABOmatch              object
CMVstatus             object
DonorCMV              object
RecipientCMV          object
Disease               object
Antigen               object
Alel                  object
extcGvHD              object
CD3dCD34 numeric      object
```

```
CD3dkgx10d8 numeric    object
Rbodymass numeric      object
dtype: object
```

U tabeli se nalaze vrednosti koje nisu samo numeričke (int / float).
Potrebno je proveriti da li neke kolone koje imaju više tipova podataka (njihov tip je object) sadrže i nepostojeće vrednosti (obeležene su sa "?" npr). Kolone koje sadrže neprihvaćene vrednosti su u podacima obeležene zelenom bojom.

Sređivanje podataka Ni u jednoj klasi obeležja ne nedostaje veliki broj vrednosti. One nevalidne se u slučaju kategoričke vrednosti menjaju modom, a u slučaju kontinualne medijanom.

[ ]:
```
data.replace('?', np.nan, inplace=True)
missing_per_rows = data.isnull().sum(axis=1)

cols_mod = ["RecipientABO", "RecipientRh", "ABOmatch", "CMVstatus", "DonorCMV",␣
 ↪"RecipientCMV", "Antigen", "Alel", "extcGvHD"]
data[cols_mod] = data[cols_mod].fillna(pd.DataFrame.mode(data[cols_mod]).
 ↪iloc[0])
data[cols_mod] = data[cols_mod].astype('int64')

cols_median = ["CD3dCD34 numeric", "CD3dkgx10d8 numeric", "Rbodymass numeric"]
data[cols_median] = data[cols_median].fillna(data[cols_median].median())
data[cols_median] = data[cols_median].astype('float64')
```

Svi podaci koji su brojevi a predstavljeni su kao string u bazi se konvertuju u odgovarajući tip podataka

[ ]:
```
data['RecipientABO'] = data['RecipientABO'].astype('int64')
data['RecipientRh'] = data['RecipientRh'].astype('int64')
data['ABOmatch'] = data['ABOmatch'].astype('int64')
data['CMVstatus'] = data['CMVstatus'].astype('int64')
data['DonorCMV'] = data['DonorCMV'].astype('int64')
data['RecipientCMV'] = data['RecipientCMV'].astype('int64')
data['Antigen'] = data['Antigen'].astype('int64')
data['Alel'] = data['Alel'].astype('int64')
data['extcGvHD'] = data['extcGvHD'].astype('int64')

data['CD3dCD34 numeric'] = data['CD3dCD34 numeric'].astype('float64')
data['CD3dkgx10d8 numeric'] = data['CD3dkgx10d8 numeric'].astype('float64')
data['Rbodymass numeric'] = data['Rbodymass numeric'].astype('float64')
```

Kodovanje kategoričkog atributa disease

[ ]:
```
print(set(data['Disease']))
data = data.replace({'Disease' : {'chronic' : 0, 'lymphoma' : 1, 'AML' : 2,␣
 ↪'nonmalignant' : 3, 'ALL': 4}})
```

```
{'ALL', 'AML', 'lymphoma', 'nonmalignant', 'chronic'}
```

[ ]:
```
# ovaj atribut potpuno nema smisla jer je direktno korelisan sa izlazom (ako je
↪neko praćen samo 100 dana, skoro sigurno je preminuo)
data.drop(['survival_time numeric'], axis = 1, inplace = True)
```

S obzirom da radimo sa binarnom klasifikacijom, nije neophodno grupisati klase.

3 2. Izdvajanje obeležja

3.1 2.1. Selekcija obeležja

[ ]:
```
# ovde se bira samo 10 obeležja - treba proveriti sa Marijom Novičić da li se
↪biraju proizvoljna ili neka istaknuta
data_corr = data[data.columns[0 : 10]]
data_corr = data_corr.join(data['survival_status numeric'].copy())
data_corr
```

[ ]:

	Recipientgender	Stemcellsource	Donorage numeric	Donorage35	IIIV	\
0	1	1	22.830137	0	1	
1	1	0	23.342466	0	1	
2	1	0	26.394521	0	1	
3	0	0	39.684932	1	1	
4	0	1	33.358904	0	0	
..	…	…	…	…	…	
182	1	1	37.575342	1	1	
183	0	1	22.895890	0	0	
184	0	1	27.347945	0	1	
185	1	1	27.780822	0	1	
186	1	1	55.553425	1	1	

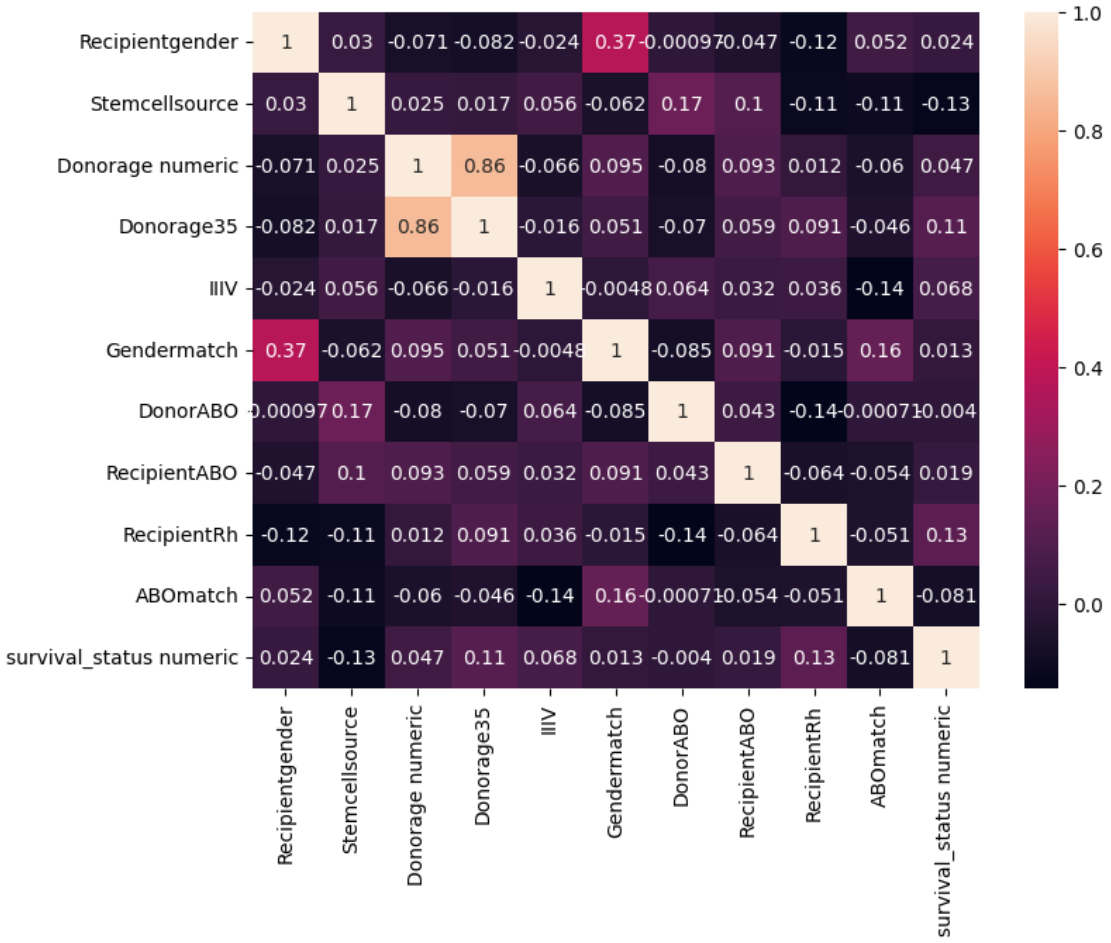
	Gendermatch	DonorABO	RecipientABO	RecipientRh	ABOmatch	\
0	0	1	1	1	0	
1	0	-1	-1	1	0	
2	0	-1	-1	1	0	
3	0	1	2	1	1	
4	0	1	2	0	1	
..	…	…	…	…	…	
182	0	1	1	0	0	
183	0	1	0	1	1	
184	0	1	-1	1	1	
185	0	1	0	1	1	
186	0	1	2	1	1	

	survival_status numeric
0	0
1	1
2	1
3	1

```
4                          0
..                       ...
182                        1
183                        1
184                        0
185                        1
186                        0

[187 rows x 11 columns]
```

Pirsonov koeficijent korelacije je prikladan za procenu linearnih zavisnosti između kontinualnih tipova obeležja, dok je Spirmanov koeficijent korelacije pogodan za procenu monotonih odnosa (koji nisu nužno linearni) između obeležja, odnosno promenljivih koje mogu biti numeričke ili kategoričke. S obzirom da pri radu sa našom bazom baratamo i sa kategoričkim obeležjima, odlučujemo se za Spirmanov koeficijent korelacije.

[ ]:
```
spearman_R = data_corr.corr(method='spearman')
plt.figure(figsize=(6.4 * 1.3, 4.8 * 1.3))
sns.heatmap(spearman_R, annot=True)
plt.show()
```



[ ]:
```
def calculateInfoD(col):
    unique = Counter(col)

    total = len(col)
    frequencies = np.array(list(unique.values()), dtype=np.int64)

    p_all = frequencies / total
    infoD = -np.sum(np.multiply(p_all, np.log2(p_all)))
    return infoD
```

$$\text{Info}(D) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

$$\text{Info}(D/A) = \sum_{j=1}^{m} \frac{|D_j|}{|D|} \text{Info}(D_j)$$

$$IG = \text{Info}(D) - \text{Info}(D/A)$$

[ ]:
```
def analysisIG(data_corr, infoD, klasa):
    IG = dict()

    for feature_idx in range(data_corr.shape[1]-1):
        col_values = data_corr.iloc[:, feature_idx]
        f = np.unique(col_values)
        infoDA = 0
        for i in f:
            temp = klasa[col_values == i]

            infoDi = calculateInfoD(temp)
            Di = sum(col_values == i)
            D = len(col_values)

            infoDA += Di * infoDi / D

        IG[feature_idx] = infoD - infoDA

        print('IG' + str(feature_idx) + ' = ' 'IG(' + data_corr.
 ↪columns[feature_idx] + ') = ' + str(infoD - infoDA))
        print('------')

    IGsorted = dict(sorted(IG.items(), key=lambda item: item[1]))
    print('Sortirano IG = \n', IGsorted)

    return IGsorted
```

[ ]:
```
klasa = data_corr.iloc[:,-1]
infoD = calculateInfoD(klasa)

IGsorted = analysisIG(data_corr, infoD, klasa)
```

```
IG0 = IG(Recipientgender) = 0.00041248804242211
------
IG1 = IG(Stemcellsource) = 0.011480699921597504
------
IG2 = IG(Donorage numeric) = 0.9940302114769565
------
IG3 = IG(Donorage35) = 0.00937784655669216
------
IG4 = IG(IIIV) = 0.0033176265497799617
------
IG5 = IG(Gendermatch) = 0.00012104129368917249
------
IG6 = IG(DonorABO) = 0.01199330323187775
------
IG7 = IG(RecipientABO) = 0.0026128131980167613
------
IG8 = IG(RecipientRh) = 0.01269866843063927
------
IG9 = IG(ABOmatch) = 0.004676509383316874
------
Sortirano IG =
 {5: 0.00012104129368917249, 0: 0.00041248804242211, 7: 0.0026128131980167613,
4: 0.0033176265497799617, 9: 0.004676509383316874, 3: 0.00937784655669216, 1:
0.011480699921597504, 6: 0.01199330323187775, 8: 0.01269866843063927, 2:
0.9940302114769565}
```

Ono što možemo zaključiti na osnovu vrednosti IG za 10 odabranih obeležja jeste da uzrast donora nosi najveću informaciju, dok preklapanje polova predstavlja najmanje značajan podatak za određivanje klase.

3.1.1 Analiza nad svim karakteristikama

[ ]:
```
data_corr = data.drop('survival_status numeric', axis = 1)
data_corr = data_corr.join(data['survival_status numeric'].copy())
```

[ ]:
```
klasa = data_corr.iloc[:,-1]
infoD = calculateInfoD(klasa)

IGsorted = analysisIG(data_corr, infoD, klasa)
IGsorted = dict(sorted(IGsorted.items(), key=lambda item: item[1],␣
 ↪reverse=True))
```

```
IG0 = IG(Recipientgender) = 0.00041248804242211
------
```

```
IG1 = IG(Stemcellsource) = 0.011480699921597504
------
IG2 = IG(Donorage numeric) = 0.9940302114769565
------
IG3 = IG(Donorage35) = 0.00937784655669216
------
IG4 = IG(IIIV) = 0.0033176265497799617
------
IG5 = IG(Gendermatch) = 0.00012104129368917249
------
IG6 = IG(DonorABO) = 0.01199330323187775
------
IG7 = IG(RecipientABO) = 0.0026128131980167613
------
IG8 = IG(RecipientRh) = 0.01269866843063927
------
IG9 = IG(ABOmatch) = 0.004676509383316874
------
IG10 = IG(CMVstatus) = 0.00805829638734823
------
IG11 = IG(DonorCMV) = 0.002620124005733193
------
IG12 = IG(RecipientCMV) = 0.0035498481808541316
------
IG13 = IG(Disease) = 0.058865886311242344
------
IG14 = IG(Riskgroup) = 0.015736015242410084
------
IG15 = IG(Txpostrelapse) = 0.009675425877970212
------
IG16 = IG(Diseasegroup) = 0.0038445765348088523
------
IG17 = IG(HLAmatch) = 0.002780467756482663
------
IG18 = IG(HLAmismatch) = 4.8563374402621484e-05
------
IG19 = IG(Antigen) = 0.0036059933515905085
------
IG20 = IG(Alel) = 0.00752287741660973
------
IG21 = IG(HLAgrI) = 0.006875287775597427
------
IG22 = IG(Recipientage numeric) = 0.6617362783891618
------
IG23 = IG(Recipientage10) = 0.016411192719839707
------
IG24 = IG(Recipientageint) = 0.015326114558630755
------
```

```
IG25 = IG(Relapse) = 0.07236686758423094
------
IG26 = IG(aGvHDIIIIV) = 0.011453931369030834
------
IG27 = IG(extcGvHD) = 0.01190761180607447
------
IG28 = IG(CD34kgx10d6 numeric) = 0.9619446499796303
------
IG29 = IG(CD3dCD34 numeric) = 0.9747273212393265
------
IG30 = IG(CD3dkgx10d8 numeric) = 0.8770553292260088
------
IG31 = IG(Rbodymass numeric) = 0.6776598400398968
------
IG32 = IG(ANCrecovery numeric) = 0.13474266107344945
------
IG33 = IG(PLTrecovery numeric) = 0.3952129278320009
------
IG34 = IG(time_to_aGvHD_III_IV numeric) = 0.13818269006400996
------
Sortirano IG =
 {18: 4.8563374402621484e-05, 5: 0.00012104129368917249, 0: 0.00041248804242211,
7: 0.0026128131980167613, 11: 0.002620124005733193, 17: 0.002780467756482663, 4:
0.0033176265497799617, 12: 0.0035498481808541316, 19: 0.0036059933515905085, 16:
0.0038445765348088523, 9: 0.004676509383316874, 21: 0.006875287775597427, 20:
0.00752287741660973, 10: 0.00805829638734823, 3: 0.00937784655669216, 15:
0.009675425877970212, 26: 0.011453931369030834, 1: 0.011480699921597504, 27:
0.01190761180607447, 6: 0.01199330323187775, 8: 0.01269866843063927, 24:
0.015326114558630755, 14: 0.015736015242410084, 23: 0.016411192719839707, 13:
0.058865886311242344, 25: 0.07236686758423094, 32: 0.13474266107344945, 34:
0.13818269006400996, 33: 0.3952129278320009, 22: 0.6617362783891618, 31:
0.6776598400398968, 30: 0.8770553292260088, 28: 0.9619446499796303, 29:
0.9747273212393265, 2: 0.9940302114769565}
```

[ ]:
```
# top 10 karakterisitka
list(IGsorted.keys())[0:9]
```

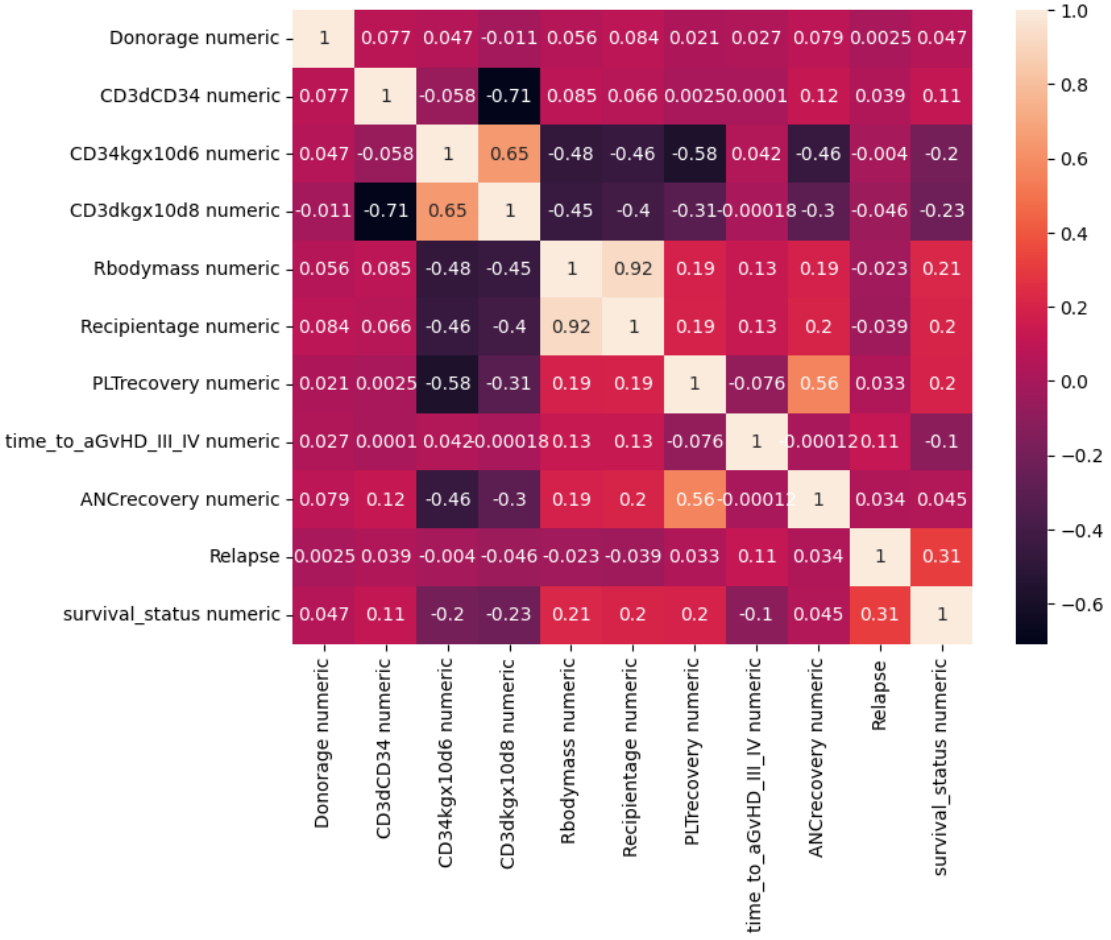
[ ]:
```
[2, 29, 28, 30, 31, 22, 33, 34, 32]
```

[ ]:
```
data_corr_10 = data_corr[data_corr.columns[list(IGsorted.keys())[0:10]]]
data_corr_10['survival_status numeric'] = data_corr['survival_status numeric'].
  ↪values
```

```
C:\Users\jn200100d\AppData\Local\Temp\ipykernel_15088\1463498987.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_corr_10['survival_status numeric'] = data_corr['survival_status
numeric'].values
```

[ ]:
```
spearman_R = data_corr_10 .corr(method='spearman')
plt.figure(figsize=(6.4 * 1.3, 4.8 * 1.3))
sns.heatmap(spearman_R, annot=True)
plt.show()
```



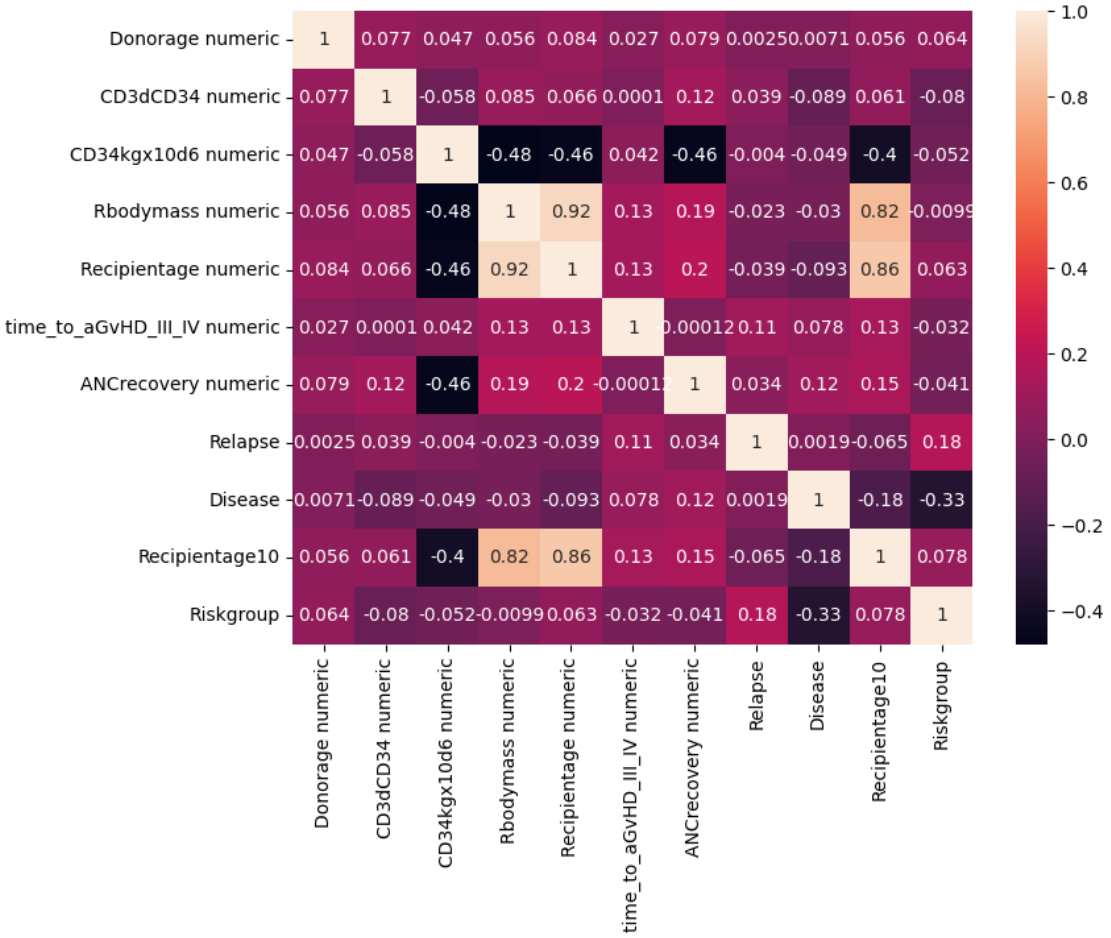
Vidi se da je međusobna korelisanost fičera CD3dkgx10d8 i CD3dCD34, PLTrecovery i CD34dkgx10d6, CD34dkgx10d6 i CD3dkgx10d8, kao i ANCrecovery i PLTrecovery veoma velika, tako da je potrebno izbaciti neke od njih.

[ ]:
```
data_corr_10 = data_corr[data_corr.columns[list(IGsorted.keys())[0:13]]]
data_corr_10.drop('CD3dkgx10d8 numeric', axis = 1, inplace = True)
data_corr_10.drop('PLTrecovery numeric', axis = 1, inplace = True)
```

```
C:\Users\jn200100d\AppData\Local\Temp\ipykernel_15088\4105094829.py:2:
```

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_corr_10.drop('CD3dkgx10d8 numeric', axis = 1, inplace = True)
C:\Users\jn200100d\AppData\Local\Temp\ipykernel_15088\4105094829.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_corr_10.drop('PLTrecovery numeric', axis = 1, inplace = True)
```

[ ]:
```
spearman_R = data_corr_10.corr(method='spearman')
plt.figure(figsize=(6.4 * 1.3, 4.8 * 1.3))
sns.heatmap(spearman_R, annot=True)
plt.show()
```



Ovakva korelacija i informaciona dobit je korektna za 10 najboljih karakteristika koje bi se kasnije razmatrale.

3.2 2.2. Ekstrakcija obeležja

[ ]:
```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

[ ]:
```
X = data.drop('survival_status numeric', axis=1)
D = data['survival_status numeric']
```

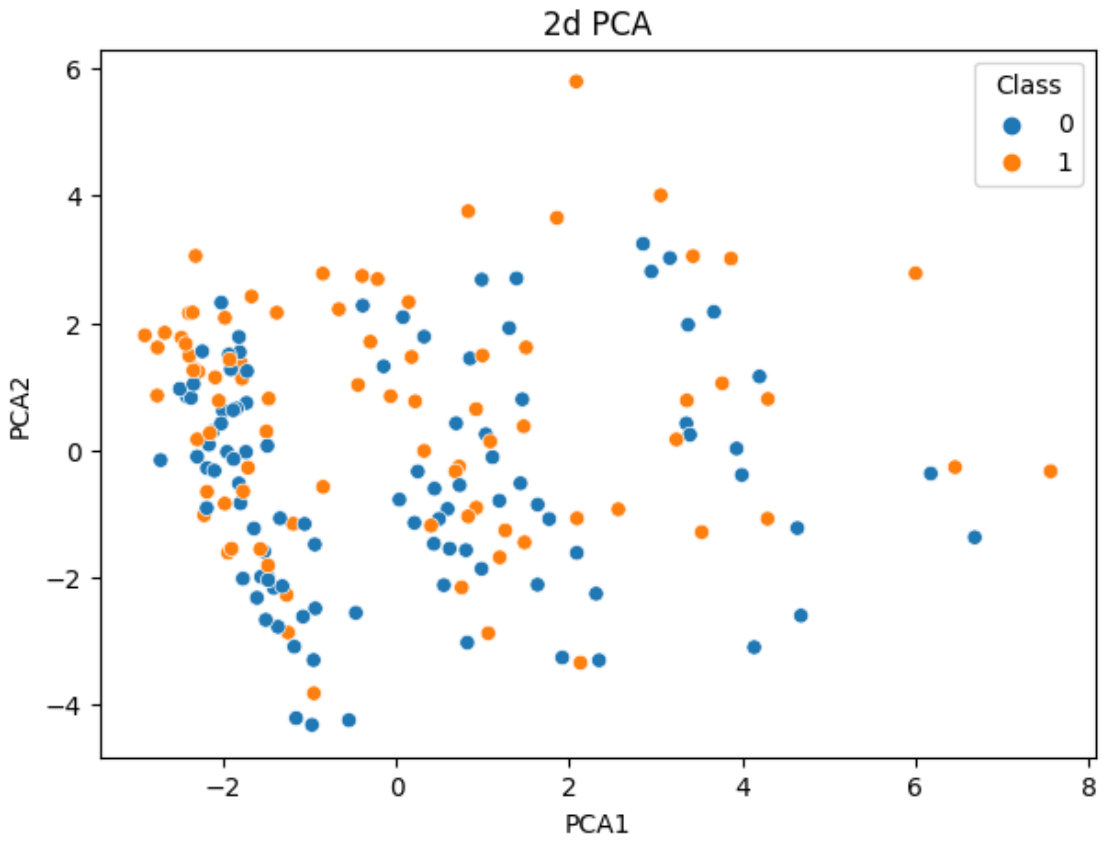
[ ]:
```
def normalize(X):

    # za analizu sa 10 obeležja
    # X_norm = data_corr_10
    X_norm = X
    X_mean = np.mean(X, axis = 0)
    X_max = np.max(X, axis = 0)
    X_std = np.std(X, axis = 0)

    cols = ['Donorage numeric', 'Recipientage numeric', 'CD34kgx10d6 numeric',␣
↪'CD3dCD34 numeric', 'CD3dkgx10d8 numeric', 'Rbodymass numeric', 'ANCrecovery␣
↪numeric', 'PLTrecovery numeric', 'time_to_aGvHD_III_IV numeric']
    #X_norm[cols] = (X[cols] - X_mean[cols]) / X_max[cols]

    X_norm[cols] = (X[cols] - X_mean[cols]) / X_std[cols]

    return X_norm
```

Nema smisla koristiti LDA metodu na sužavanje skupa karakteristika na 2 ili 3 dimenzije jer je maksimalan broj karakteristika koji se može dobiti = min(broj_karakteristika, broj_klasa - 1) = min(36, 2 - 1) = min(36, 1) = 1 => jednodimenzionalna projekcija.

Redukcija PCA na dve dimenzije

[ ]:
```
pca = PCA(n_components = 2)
X_norm = normalize(X)
data_pca_fcn = pd.DataFrame(pca.fit_transform(X_norm))

data_pca_fcn = pd.concat([data_pca_fcn,D], axis = 1)
data_pca_fcn.columns = ['PCA1', 'PCA2', 'Class']
```

[ ]:
```
plt.figure(figsize=(7, 5))
sns.scatterplot(data = data_pca_fcn, x = 'PCA1', y = 'PCA2', hue = 'Class')
plt.title('2d PCA')
plt.show()
```

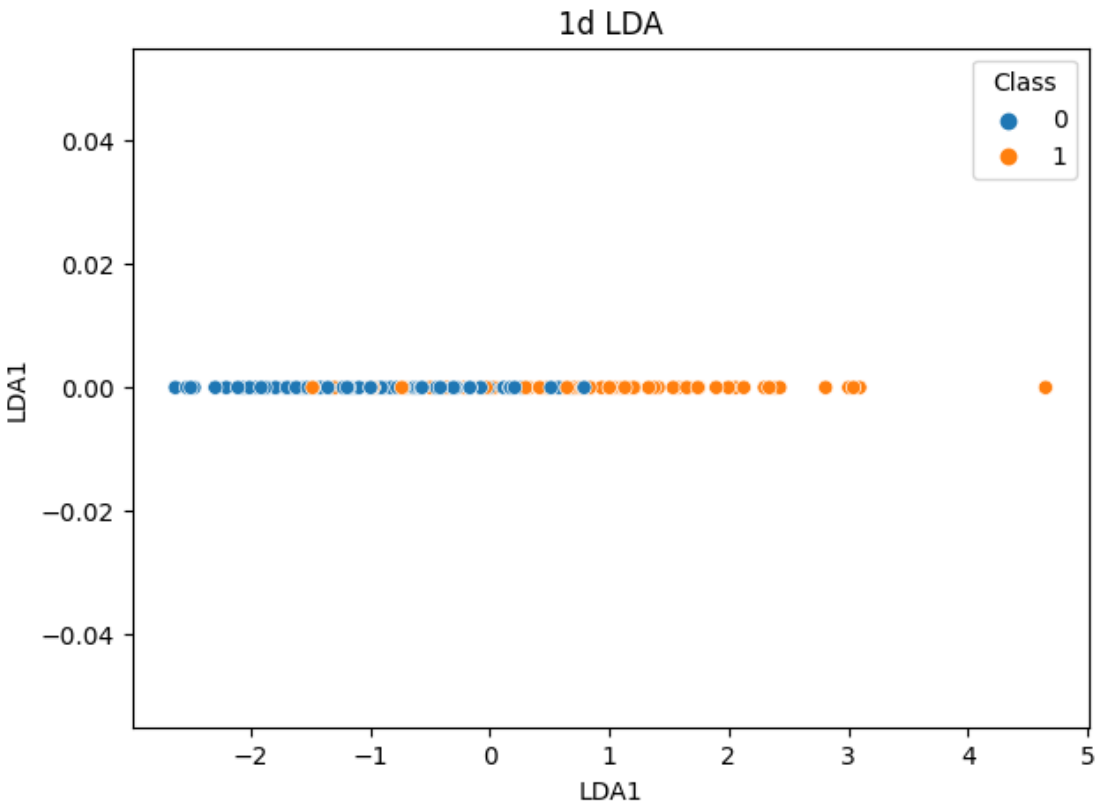


Uz drugačiju normalizaciju podataka, dobijaju se malo drugačiji rezultati, ali ništa što bi dalo uočljiviju separabilnost klasa.

Redukcija LDA na jednu dimenziju

[ ]:
```
lda = LDA(solver = 'eigen', n_components = 1)
data_lda_fcd = pd.DataFrame(lda.fit_transform(X_norm, D))

data_lda_fcd = pd.concat((data_lda_fcd, D), axis = 1)
data_lda_fcd.columns = ['LDA1', 'Class']
```

[ ]:
```
plt.figure(figsize = (7, 5))
sns.scatterplot(x = data_lda_fcd['LDA1'], y = [0]*data_lda_fcd['LDA1'], hue =␣
↪data_lda_fcd['Class'])
plt.title('1d LDA')
plt.show()
```



Komentar Pošto je ova baza podataka podeljena na dve klase - preživeli i preminuli, potrebno je uporediti performanse 1d LDA i 2d PCA metode.
Sa grafika se može zaključiti da mnogo bolje rezultate daje 1d LDA metoda. Ona se koristi kada nam je potrebno da maksimizujemo separabilnost klasa, oslanjajući se na već postojeće labele; dok je PCA nenadgledana metoda koja se bazira na pronalasku maksimalne varijanse svih podataka u bazi. Ona potencijalno može biti pogodnija ukoliko imamo manji broj odbiraka po klasi, što trenutno nije slučaj.
Grafici distribucije odbiraka po klasama su prikazani u prethodnim koracima u zavisnosti od korišćene metode.

Analiza PCA i LDA nad setom od 10 najboljih karakterisitka (postupak biranja je opisan u prethodnom koraku) dovodi do vidnog (ali ne i dovoljno dobrog) poboljšanja PCA klasifikatora, dok LDA daje relativno slične rezultate (kao za ceo skup podataka).

4 3. Projektovanje klasifikatora

4.1 3.1. Parametarska klasifikacija

[ ]:
```
X_norm_lda = X_norm.values
class_lda = D.values
```

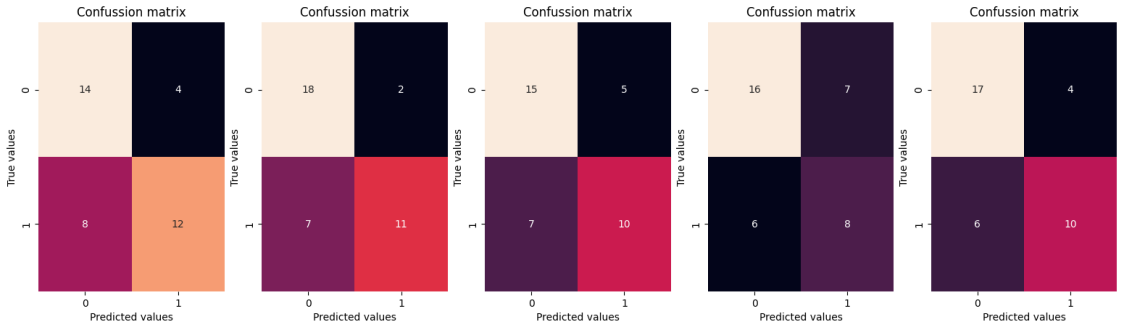
[ ]:
```
def linear_classifier(X_train_lda, y_train):
    # Izdvajanje odbiraka klase 0 i odgovarajuceg broja odbiraka koji pripadaju␣
 ↪toj klasi
    x_c0 = np.array(X_train_lda[y_train == 0])
    n_0 = np.sum(y_train == 0)
    x_c0 = x_c0.reshape(1, n_0)

    # Izdvajanje odbiraka klase 1 i odgovarajuceg broja odbiraka koji pripadaju␣
 ↪toj klasi
    x_c1 = np.array(X_train_lda[y_train == 1])
    n_1 = np.sum(y_train == 1)
    x_c1 = x_c1.reshape(1, n_1)

    # Kreiranje matrica Z1 i Z2 i pakovanje u zajednicku matricu U
    Z1 = np.matrix(np.append(-x_c0, -np.ones((1, n_0)), axis=0))
    Z2 = np.matrix(np.append(x_c1, np.ones((1, n_1)), axis=0))
    U = np.append(Z1, Z2, axis=1).T

    # Primenom metoda trazenja pseudoinverza nalazimo koeficijente W
    G = np.ones((y_train.shape[0], 1))
    W = (U.T * U) ** (-1) * U.T * G
    W = np.array(np.asarray(W).T[0])

    return W
```

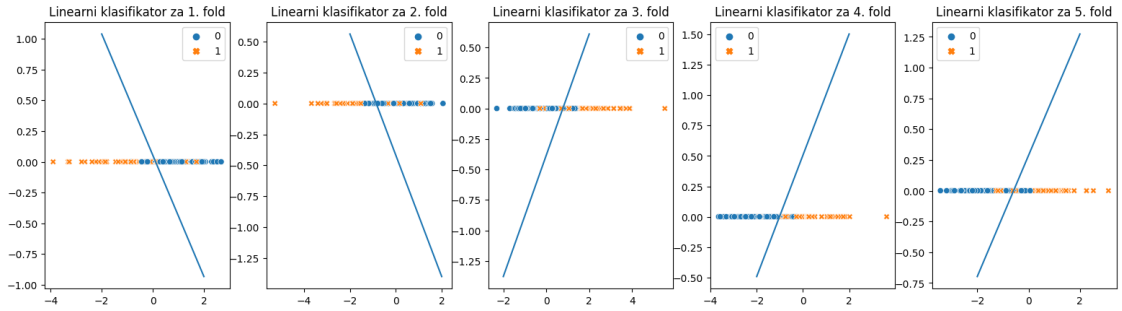
[ ]:
```
def linear_classifier_predict(W, X_test_lda):
    x = np.array(X_test_lda)

    U_test = np.matrix(np.append(x, np.ones((X_test_lda.shape[0], 1)), axis=1)).
 ↪T
    y_pred = W * U_test
    y_pred = np.asarray(y_pred.T) > 0

    return y_pred
```

[ ]:
```
import sklearn
kf = KFold(n_splits=5, shuffle=True)

fold = 1
```

```
acc = []



# iteriramo kroz definisane foldove
for train_idx, val_idx in kf.split(X_norm_lda):

    X_train, X_test, y_train, y_test = X_norm_lda[train_idx],␣
 ↪X_norm_lda[val_idx], class_lda[train_idx], class_lda[val_idx]

    # Odredimo matricu transformacije LDA metode za obucavajuci skup i␣
 ↪primenjujemo je na skup za testiranje
    X_train_lda = lda.fit_transform(X_train, y_train)
    X_test_lda = lda.transform(X_test)

    # Odredjujemo parametre linearne klasifikacije i radimo predikciju vrednosti
    W = linear_classifier(X_train_lda, y_train)

    y_pred = linear_classifier_predict(W, X_test_lda)
    y_test = y_test.reshape(y_test.shape[0], 1)

    # Evaluacija klasifikatora
    acc.append(sklearn.metrics.accuracy_score(y_pred, y_test))


    plt.figure(1, figsize=(6.4 * 3, 4.8))
    plt.subplot(1, 5, fold)
    cm = confusion_matrix(y_test, np.asarray(y_pred, dtype=int))
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False)
    plt.title("Confussion matrix")
    plt.xlabel("Predicted values")
    plt.ylabel("True values")


    plt.figure(2, figsize=(6.4 * 3, 4.8))
    plt.subplot(1, 5, fold)
    sns.scatterplot(x = X_train_lda[:, 0], y = [0]*X_train_lda[:, 0], hue =␣
 ↪y_train, style=y_train)
    plt.plot(np.linspace(-2, 2), W[0] * np.linspace(-2, 2)+W[1])
    plt.title('Linearni klasifikator za ' + str(fold) + '. fold')


    fold += 1

print('Srednja tacnost: ', np.mean(acc))
```

```
Srednja tacnost:  0.7002844950213372
```





Vidimo da je linearni klasifikator na bazi zeljenog izlaza pogodan za podatke transformisane LDA metodom i daje oko 70% tačnosti. Za γ vektor su korišćene sve jedinice. Eksperimentalnim putem je pokazano da drugačije težine (drugačiji elementi vektora γ) ne doprinose poboljšanju tačnosti.

4.2 3.2 Neparametarska klasifikacija

Podaci koje analiziramo sastoje se od 187 ispitanika sa 37 karakterističnih obeležja.
Za takvu veličinu baze, KNN bi potencijalno bio bolji izbor jer se stabla odlučivanja mogu preobučiti ili postati previše kompleksna. Međutim, dimenzionalnost baze (37 karakteristika) potencijalno može biti problem za KNN klasifikator. Takođe, kategoričke vrednosti su problematične za KNN jer se on osladnja na euklidsku distancu. Stabla odlučivanja su manje osetljive na outlier-e koje smo primetili kroz inicijalnu analizu podataka. Klase su prilično dobro izbalansirane, sa 85 preživelih osoba i 102 nesrećna slučaja. Tako da izbalansiranost takođe neće predstavljati problem i oba klasifikatora dolaze u obzir.

Iz prethodne diskusije zaključuje se da oba klasifikatora imaju potencijala za ovu bazu i problematiku, ali ćemo stablu odlučivanja dati prednost pri problemu klasifikacije.

Kao metrika za dobijanje tačnosti što nezavisnije od raspodele podataka koristiće se krosvalidacija (leave-one-out).

[ ]:
```
data['survival_status numeric'].value_counts()
```

[ ]:
```
0    102
1     85
Name: survival_status numeric, dtype: int64
```

Priprema podataka za klasifikaciju (podela na skupove za treniranje i testiranje)

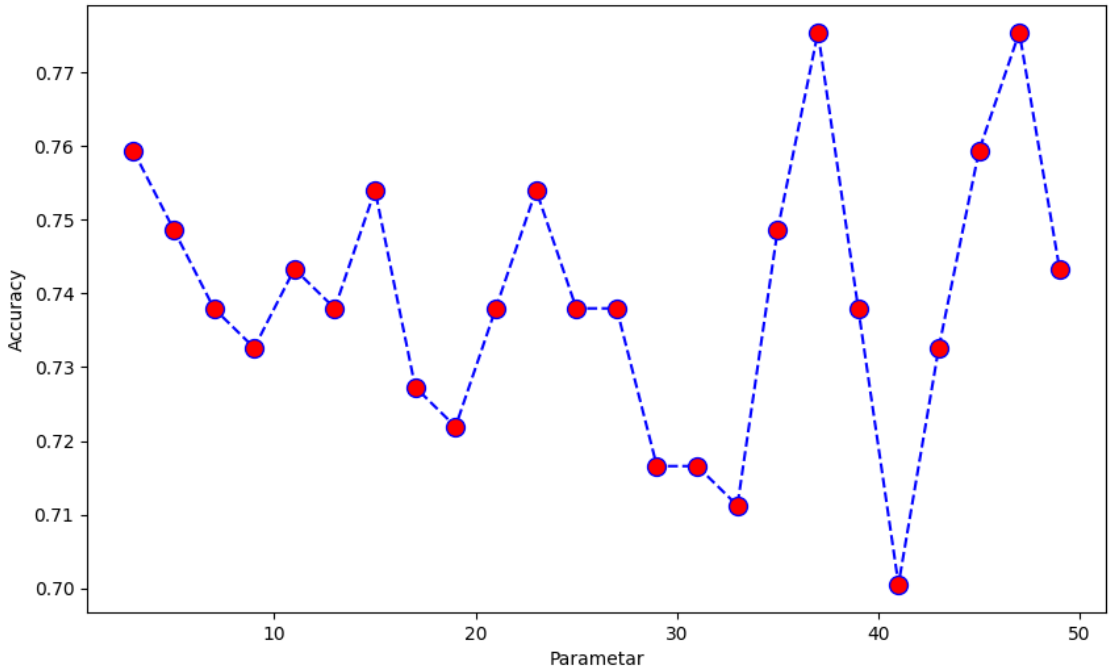
[ ]:
```
def prepare_data(data, person):

    # Uzimanje jednog pacijenta za treniranje
    X_test = data.iloc[[person]].drop('survival_status numeric', axis = 1).
 ↪reset_index(drop=True)
    y_test = list(data.iloc[[person]]['survival_status numeric'])

    # Mešanje podataka pri kome se indeksi ne gube
    data_shuff = data.sample(frac = 1)

    # Brisanje istog pacijenta i preuređivanje indeksa
    X_train = data_shuff.drop(person).drop('survival_status numeric', axis = 1).
 ↪reset_index(drop=True)
    y_train = list(data_shuff.drop(person)['survival_status numeric'])

    # Povratne vrednosti podataka za treniranje i testiranje, kao i njihovih␣
 ↪odgovarajućih labela
    return X_train, X_test, y_train, y_test
```

[ ]:
```
num_of_participants = len(data)
```

4.2.1 Stablo odlučivanja

[ ]:
```
# Lista koja će sadržati tačnosti za različite vrednosti parametra
acc_diff_param = []

acc = 0
y_pred_optim = []
y_test_optim = []

# Iteracija za različite vrednosti parametra
for param in range(3, 50, 2):

    # Lista koja će sadržati tačnosti za različita izostavljanja pacijenta
    accuracy = []

    # Iteracija za različita izostavljanja pacijenta
    kf = KFold(n_splits=187, shuffle=True)

    for train_idx, val_idx in kf.split(data):
```

```
        # Priprema podataka za treniranje i testiranje
        x_dt = data.drop('survival_status numeric',axis=1).to_numpy()
        y_dt = data['survival_status numeric'].to_numpy()
        X_train, X_test, y_train, y_test = x_dt[train_idx], x_dt[val_idx],␣
 ↪y_dt[train_idx], y_dt[val_idx]

        # Treniranje i testiranje određenog klasifikatora
        classif = DecisionTreeClassifier(max_depth=param)
        classif.fit(X_train, y_train)

        y_pred = classif.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_pred))

        if(accuracy_score(y_test, y_pred) > acc):
            y_test_optim = y_test
            y_pred_optim = y_pred
            acc = accuracy_score(y_test, y_pred)

    acc_diff_param.append(np.mean(accuracy))

# Plotovanje zavisnosti tačnosti za različite vrednosti parametra
figure = plt.figure(figsize=(10, 6))
plt.plot(range(3, 50, 2), acc_diff_param, color='blue', linestyle='dashed',␣
 ↪marker='o',
            markerfacecolor='red', markersize=10)
plt.xlabel('Parametar')
plt.ylabel('Accuracy')
plt.show()

# Ispis maksimalne tačnosti i vrednosti parametra za koju je ona postignuta
vr = acc_diff_param.index(max(acc_diff_param)) * 2 + 3
print('Maksimalna tacnost: ', max(acc_diff_param), ', za vrednost parametra: ',␣
 ↪vr)
```




```
Maksimalna tacnost:  0.7754010695187166 , za vrednost parametra:  37
```

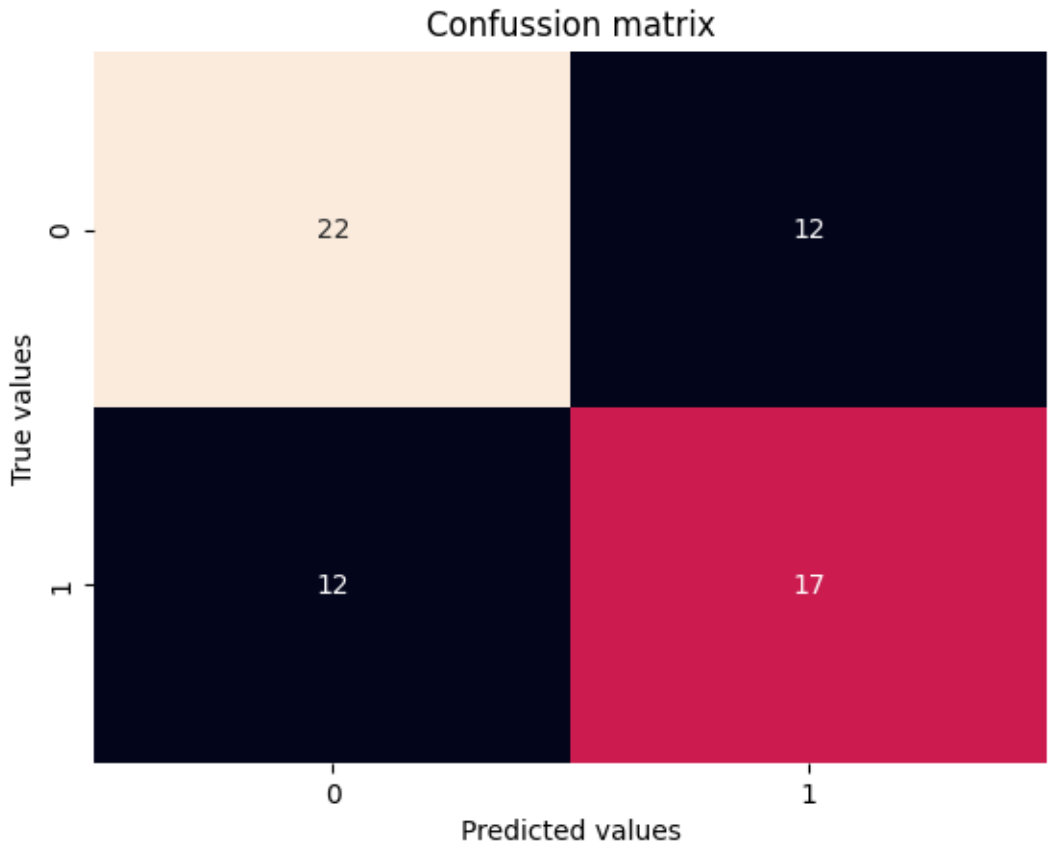
Iz priloženih kodova i analiza može se videti da stabla odlučivanja daju preko 75% tačnosti. Ponovnim pokretanjem prethodnog koda dobijamo različitu optimalnu dubinu. Iz ovoga možemo zaključiti da je za ovu problematiku pogodnije koristiti Random Forest ansambl metod. Na osnovu eksperimentisanja je pokazano da za 1000 klasifikatora (stabla odlučivanja) postižemo preko 90% tačnosti. Stoga, mislimo, da je ovaj metod najbolji za datu bazu (u odnosu na sve analizirane metode klasifikacije).

Za ovakav pristup problemu nije najpraktičnije prikazati matricu konfuzije. Kao što je i pretpostavljeno, klasifikacija sa korišćenjem leave one out krosvalidacije daje bolje rezultate. Međutim, zbog zahteva zadatka, u narednom segmentu koda prikazana je matrica konfuzije sa drugačijom validacijom (i lošijim rezultatom klasifikacije).

[ ]:
```
# Podela podataka
X_train, X_test, y_train, y_test = train_test_split(data.drop('survival_status␣
 ↪numeric', axis = 1), data['survival_status numeric'], train_size=2/3,␣
 ↪random_state=42, stratify=data['survival_status numeric'])

# Treniranje i testiranje stabla odlučivanja
classif = DecisionTreeClassifier(max_depth=33)
classif.fit(X_train, y_train)

y_pred = classif.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print('Accuracy: ', accuracy)

# Prikaz maatrice konfuzije
cm = confusion_matrix(y_test, y_pred)
class_labels = ['Class 0', 'Class 1']
plt.figure()
sns.heatmap(cm, annot=True, fmt='g', cbar=False)
plt.title("Confussion matrix")
plt.xlabel("Predicted values")
plt.ylabel("True values")
plt.show()
```

```
Accuracy:  0.6190476190476191
```



4.3 3.3. Neuralna mreža

[ ]:
```
import tensorflow as tf

print(tf.__version__)
```

```
2.10.1
```

Definisanje klase modela neuralne mreže

```
[ ]: class MLP(tf.keras.Model):
        def __init__(self, input_size, hidden_units, output_size, dropout_rate=0,␣
     ↪l2_reg=0):
            super(MLP, self).__init__()

            # sloj za transformisanje iz matrice u "ispravljeni" niz
            self.flatten_layer = tf.keras.layers.Flatten(input_shape=(input_size, ))

            # skriveni slojevi sa relu aktivacionom funkcijom
            self.hidden_layers = []
            for units in hidden_units:
                self.hidden_layers.append(
                    tf.keras.layers.Dense(units, activation='relu',␣
     ↪kernel_regularizer=tf.keras.regularizers.l2(l2_reg))
                )

            # dropout sloj
            self.dropout_layer = tf.keras.layers.Dropout(dropout_rate)

            # izlazni sloj
            self.output_layer = tf.keras.layers.Dense(output_size,␣
     ↪activation='softmax')


        def call(self, inputs):
            x = self.flatten_layer(inputs)

            for hidden_layer in self.hidden_layers:
                x = hidden_layer(x)
                x = self.dropout_layer(x)

            output = self.output_layer(x)
            return output
```

Definisanje pomoćnih funkcija za obučavanje modela i prikaz rezultata

```
[ ]: X_norm_data= X_norm.to_numpy()
     y = D.to_numpy()

     X_train, X_val, y_train, y_val = train_test_split(X_norm_data, y, test_size=0.
      ↪3, random_state=42)
```

```
[ ]: # lr - learning rate
     # bs - batch size

     def train_model(num_epochs, hidden_units, dropout=0, early_stopping_patience=0,␣
      ↪lr=0.01, bs=8, l2=0):
```

```
    model = MLP(X_train.shape[1], hidden_units, output_size=2,␣
↪dropout_rate=dropout, l2_reg=l2)

    # koristi se Adam optimizer
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),␣
↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).
↪batch(bs)
    test_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val)).batch(bs)

    # pozivanje funkcije fit (razlikujemo slucaj sa ranim zaustavljanjem i bez)
    if early_stopping_patience:
        early_stopping = tf.keras.callbacks.
↪EarlyStopping(monitor='val_accuracy', mode='max',␣
↪patience=early_stopping_patience, verbose=1)
        with tf.device('/GPU:0'):
            history = model.fit(train_dataset, epochs=num_epochs,␣
↪batch_size=bs, validation_data=test_dataset, verbose=0,␣
↪callbacks=[early_stopping])

    else:
        with tf.device('/GPU:0'):
            history = model.fit(train_dataset, epochs=num_epochs,␣
↪batch_size=bs, validation_data=test_dataset, verbose=0)

    # predikcija na osnovu obucenog modela
    y_pred = np.argmax(model.predict(X_val, verbose=0), axis=1)

    return history, y_pred, y_val
```

4.3.1 3.3.1. Eksperimentisanje sa različitim vrednostima za learning rate i batch size

U narednom segmentu koda je uzeto da postoje 2 skrivena sloja sa po 8 i 16 ćvorova. Drugačiji izbor ne daje mnogo različite rezultate.

[ ]:
```
hidden_unit_size = [8, 16]
epochs = 30

acc_res = dict()
loss_res = dict()
val_acc_res = dict()
val_loss_res = dict()

for lr in [0.0001, 0.001, 0.01]:
    acc_res[lr] = dict()
    loss_res[lr] = dict()
```

```
    val_acc_res[lr] = dict()
    val_loss_res[lr] = dict()
    for bs in [2, 4, 8, 16, 32]:
        history, y_pred, y_val = train_model(epochs, hidden_unit_size, lr=lr,␣
↪bs=bs)

        acc_res[lr][bs] = history.history['accuracy'][-1]
        val_acc_res[lr][bs] = history.history['val_accuracy'][-1]
        loss_res[lr][bs] = history.history['loss'][-1]
        val_loss_res[lr][bs] = history.history['val_loss'][-1]
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001C0A7D4D1F0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 11 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001C0A7D088B0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

Prikaz rezultata

[ ]:
```
acc_res_df = pd.DataFrame(acc_res)
val_acc_res_df = pd.DataFrame(val_acc_res)
loss_res_df = pd.DataFrame(loss_res)
val_loss_res_df = pd.DataFrame(val_loss_res)
```

[ ]:
```
acc_res_df
```

[ ]:

	0.0001	0.0010	0.0100
2	0.653846	0.923077	0.984615
4	0.669231	0.861538	0.961538
8	0.469231	0.853846	1.000000
16	0.630769	0.769231	1.000000
32	0.476923	0.715385	0.915385

[ ]:
```
val_acc_res_df
```

[ ]:

	0.0001	0.0010	0.0100
2	0.561404	0.578947	0.543860
4	0.526316	0.631579	0.508772
8	0.421053	0.561404	0.596491
16	0.491228	0.614035	0.543860
32	0.543860	0.508772	0.631579

[ ]:
```
loss_res_df
```

[ ]:

	0.0001	0.0010	0.0100
2	0.705767	0.236154	0.030321
4	0.621954	0.358645	0.102852
8	0.740105	0.373960	0.015004
16	0.664526	0.472715	0.018070
32	0.776827	0.592350	0.201346

[ ]:
```
val_loss_res_df
```

[ ]:

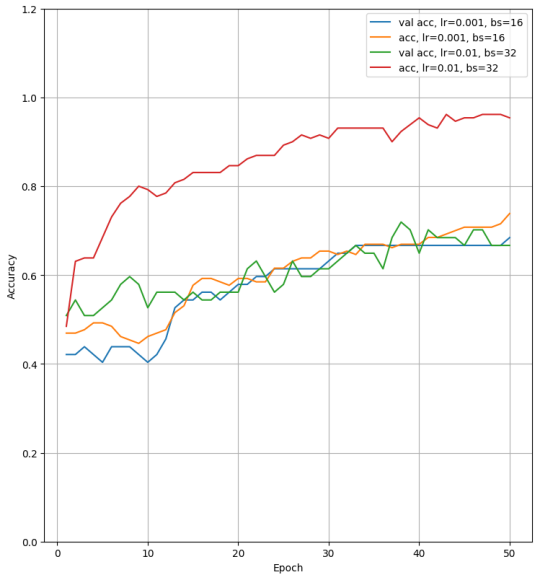
	0.0001	0.0010	0.0100
2	0.699381	0.804956	2.018985
4	0.687770	0.725004	2.036498
8	0.877889	0.669480	1.751939
16	0.660224	0.641239	2.611243
32	0.737525	0.671957	1.096876

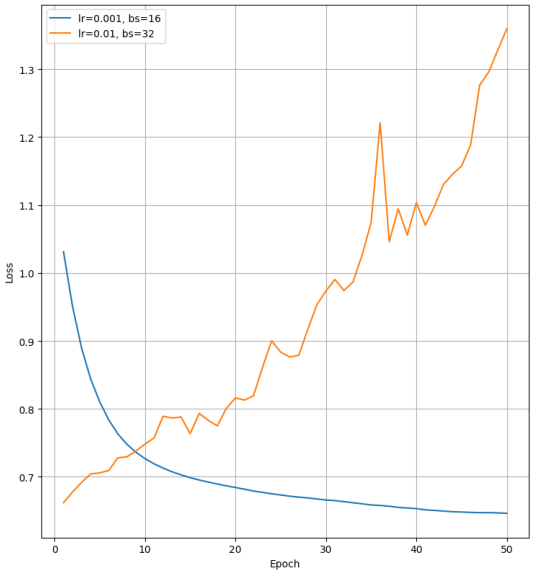
Prikaz rezultata po epohama za (lr, bs) = {(0.001, 16), (0.01, 32)}

[ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))

history1, y_pred1, y_val1 = train_model(50, [4, 8], lr=0.001, bs=16)
history2, y_pred2, y_val2 = train_model(50, [4, 8], lr=0.01, bs=32)


ax[0].plot(range(1, len(history1.history['val_accuracy'])+1), history1.
 ↪history['val_accuracy'], label='val acc, lr=0.001, bs=16')
ax[0].plot(range(1, len(history1.history['accuracy'])+1), history1.
 ↪history['accuracy'], label='acc, lr=0.001, bs=16')

ax[0].plot(range(1, len(history2.history['val_accuracy'])+1), history2.
 ↪history['val_accuracy'], label='val acc, lr=0.01, bs=32')
ax[0].plot(range(1, len(history2.history['accuracy'])+1), history2.
 ↪history['accuracy'], label='acc, lr=0.01, bs=32')

ax[0].set_xlabel('Epoch')
ax[0].set_ylabel('Accuracy')
ax[0].set_ylim(0, 1.2)
```

```
ax[0].grid()
ax[0].legend()

ax[1].plot(range(1, len(history1.history['val_loss'])+1), history1.
 ↪history['val_loss'], label='lr=0.001, bs=16')
ax[1].plot(range(1, len(history2.history['val_loss'])+1), history2.
 ↪history['val_loss'], label='lr=0.01, bs=32')

ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('Loss')
ax[1].grid()
ax[1].legend()
```

[ ]: <matplotlib.legend.Legend at 0x1c1b2569430>





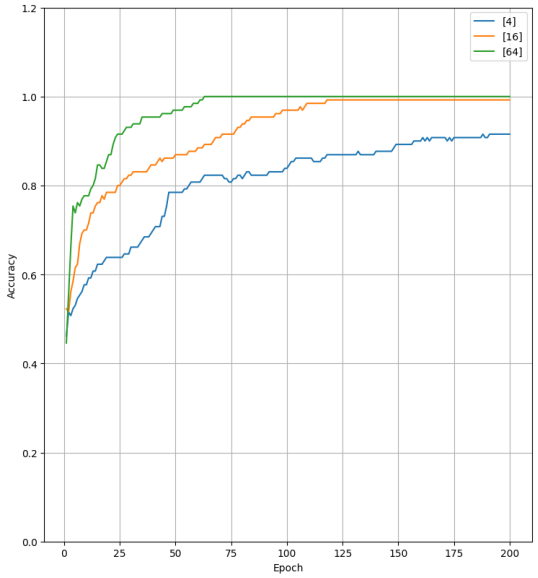
Konacan odabir vrednosti za leaning rate i batch size

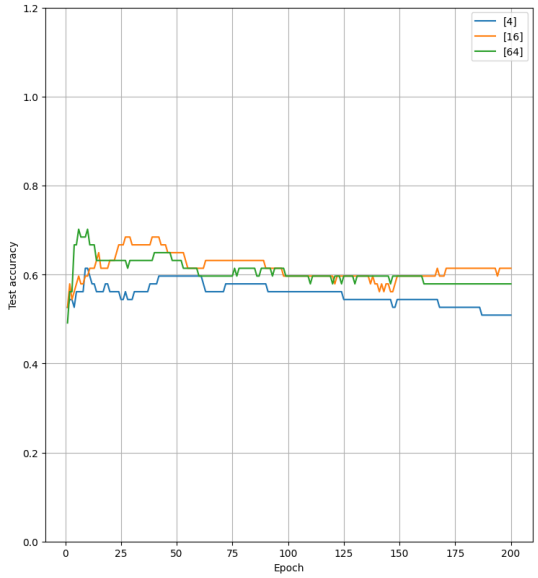
[ ]:
```
lr = 0.001
bs = 16
```

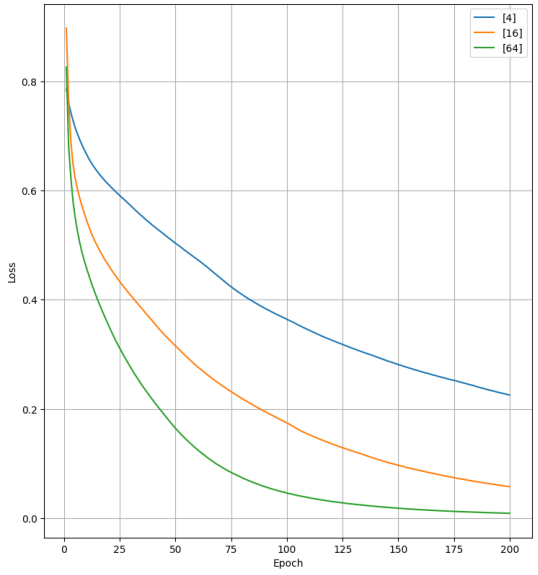
4.3.2 3.3.2. Eksperimentisanje sa razlicito definisanim skrivenim slojevima

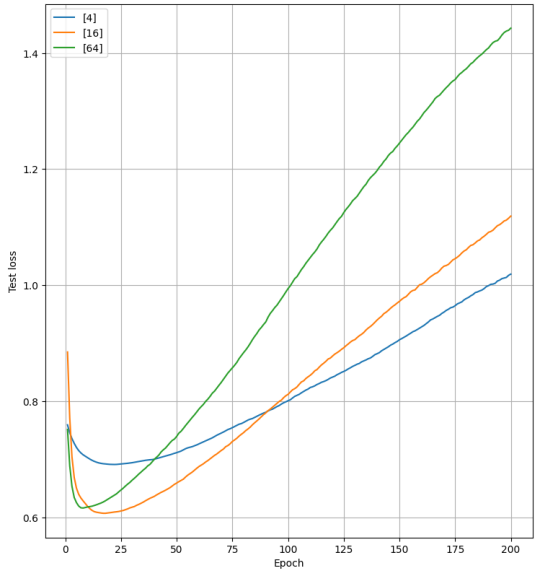
[ ]:
```
# Jedan skriveni sloj
hidden_unit_size = [[4], [16], [64]]
epochs = 200

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
```

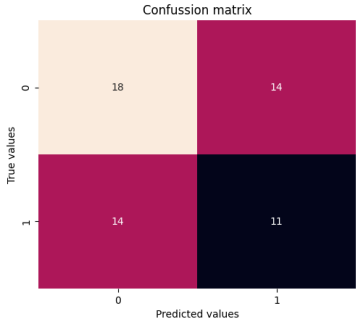
```
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0

for hus in hidden_unit_size:
    history, y_pred, y_val = train_model(epochs, hus, lr=lr, bs=bs)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=hus)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=hus)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=hus)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
 ↪history['val_loss'], label=hus)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```

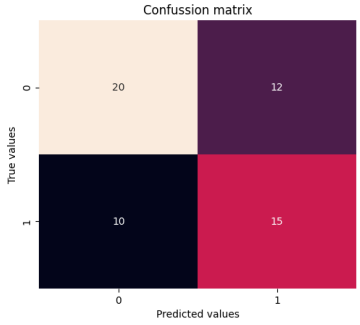

[4]
[16]
[64]
Accuracy
Epoch

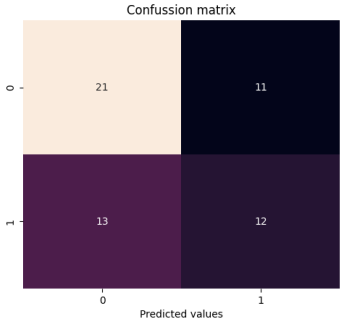

[4]
[16]
[64]
Test accuracy
Epoch


[4]
[16]
[64]
Loss
Epoch


[4]
[16]
[64]
Test loss
Epoch

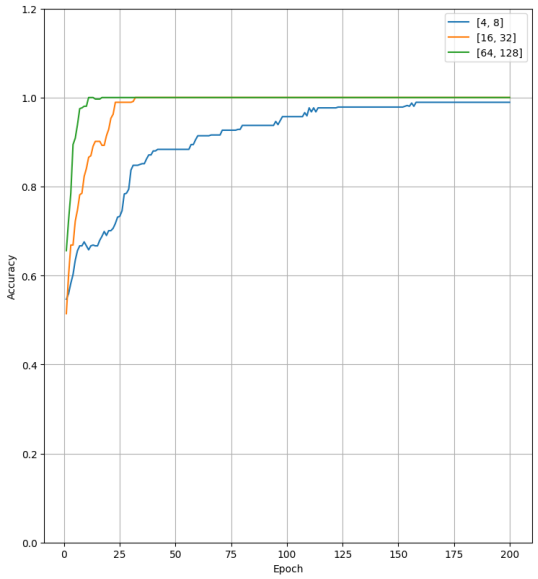

Confussion matrix
18
14
14
11
True values
Predicted values

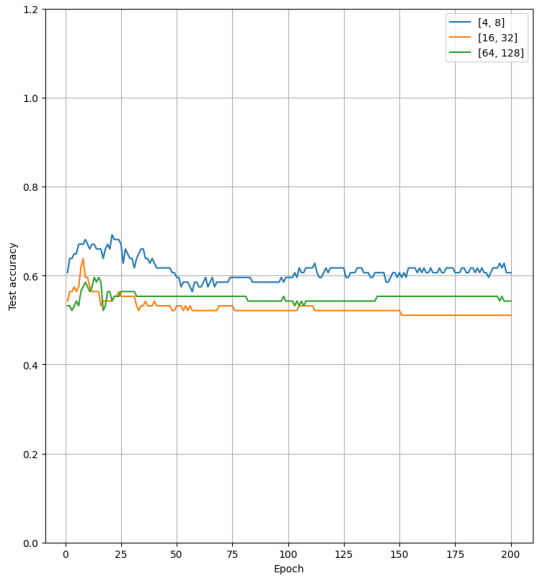

Confussion matrix
20
12
10
15
True values
Predicted values

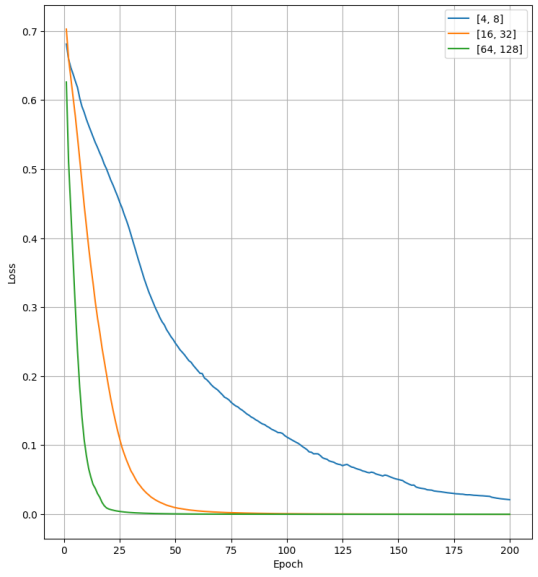

Confussion matrix
21
11
13
12
True values
Predicted values

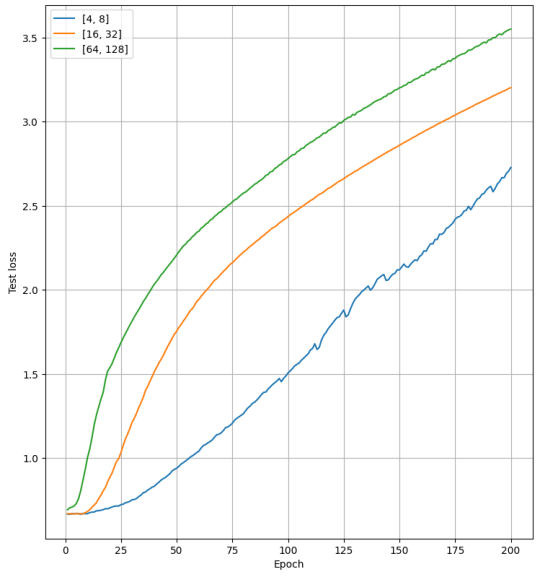
[ ]:
```
# Dva skrivena sloja
hidden_unit_size = [[4, 8], [16, 32], [64, 128]]
epochs = 200

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0

for hus in hidden_unit_size:
    history, y_pred, y_val = train_model(epochs, hus, lr=lr, bs=bs)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=hus)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=hus)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=hus)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
 ↪history['val_loss'], label=hus)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()
```

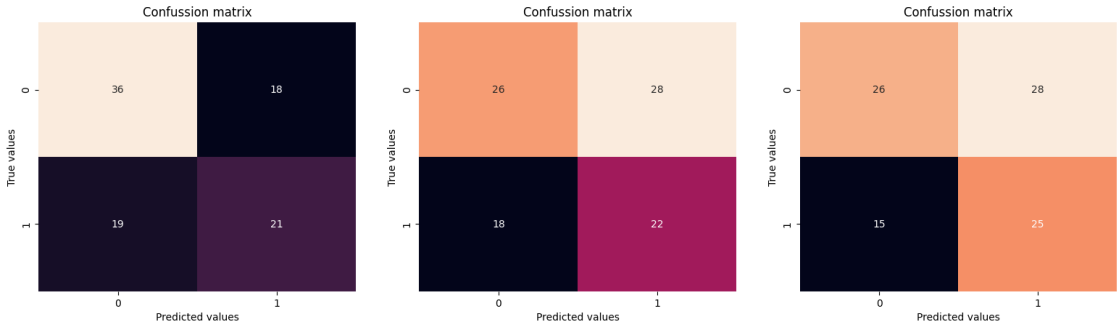
```
    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```





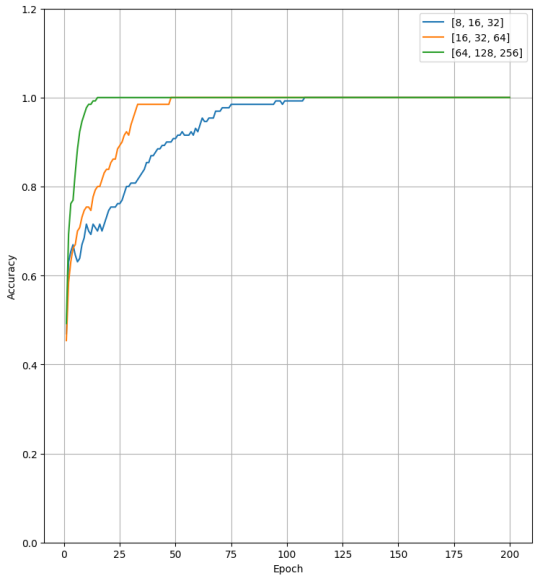


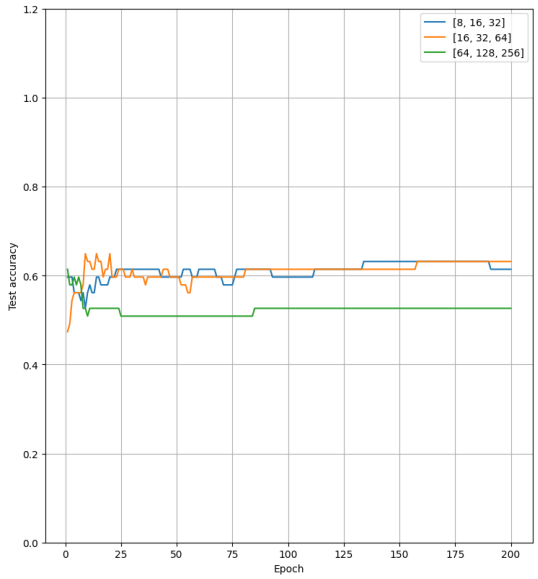


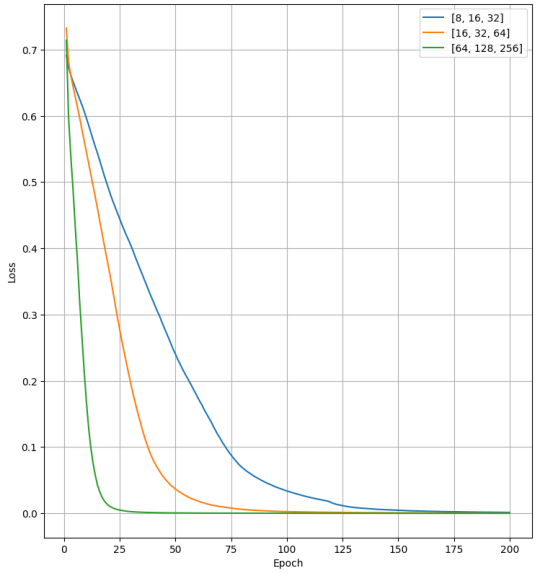


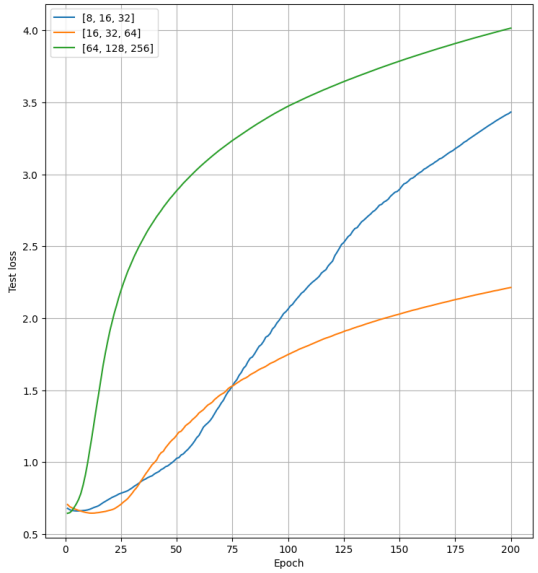
[ ]:
```
# Tri skrivena sloja
hidden_unit_size = [[8, 16, 32], [16, 32, 64], [64, 128, 256]]
epochs = 200

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0

for hus in hidden_unit_size:
    history, y_pred, y_val = train_model(epochs, hus, lr=lr, bs=bs)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=hus)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=hus)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=hus)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
```

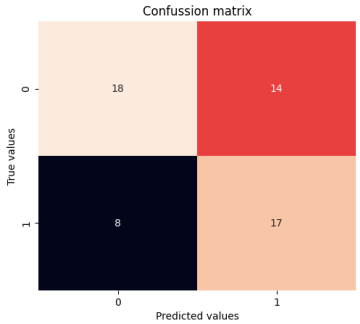
```
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
↪history['val_loss'], label=hus)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```

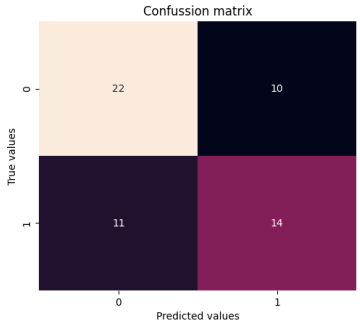


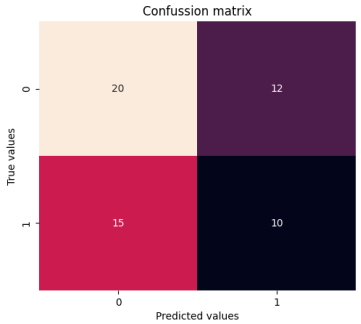










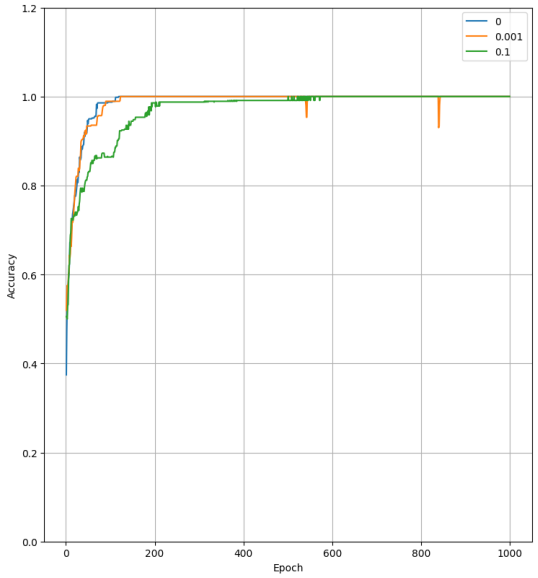


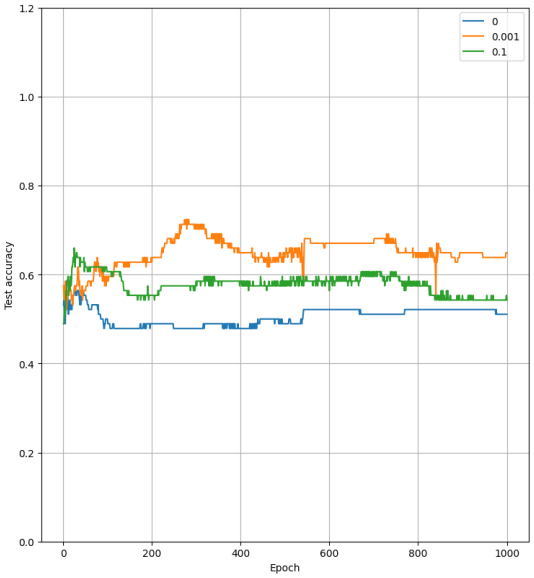
Na osnovu prethodno izvedenih eksperimenata, vidimo da ima smisla nastaviti rad sa jednom od dve opcije: mreža sa jednim skrivenim slojem (16 neurona), ili mreža sa dva skrivena sloja (4/8 neurona). Mreža sa 2 skrivena sloja ispoljava veće preprilagođavanje.

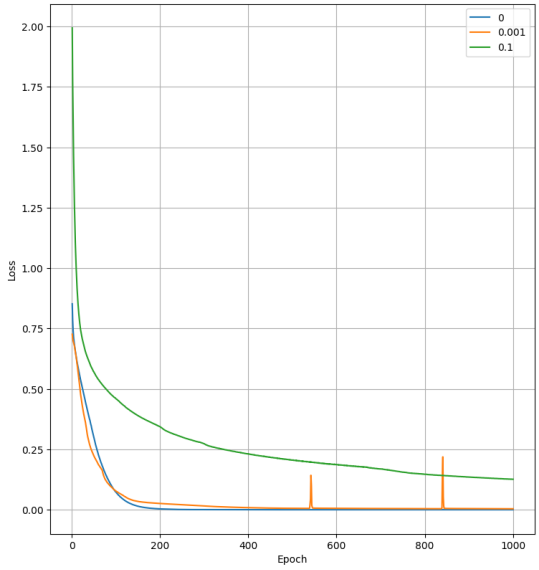
4.3.3 3.3.3. Eksperimentisanje sa dodavanjem L2 regularizacije

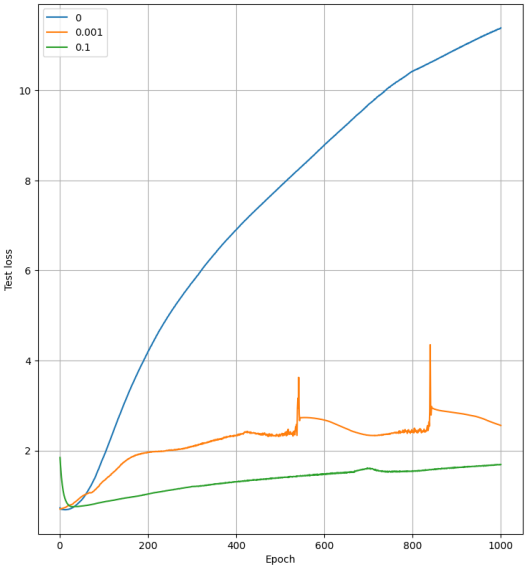
[ ]:
```
l2_reg = [0, 0.001, 0.1]
epochs = 1000

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0
```

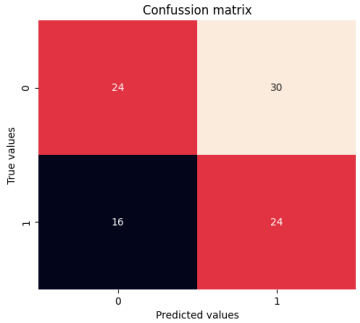
```
for l2 in l2_reg:
    history, y_pred, y_val = train_model(epochs, [4, 8], lr=lr, bs=bs, l2=l2)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=l2)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=l2)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=l2)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
 ↪history['val_loss'], label=l2)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```

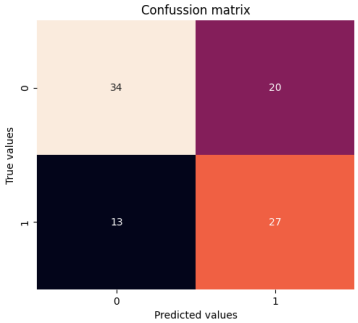

0
0.001
0.1
Accuracy
Epoch

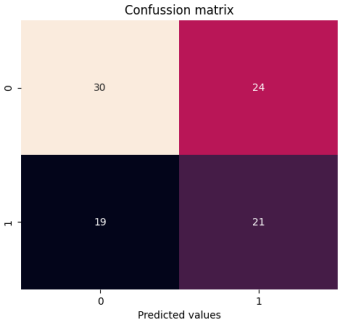

0
0.001
0.1
Test accuracy
Epoch


0
0.001
0.1
Loss
Epoch


0
0.001
0.1
Test loss
Epoch


Confussion matrix
24
30
16
24
True values
Predicted values


Confussion matrix
34
20
13
27
True values
Predicted values

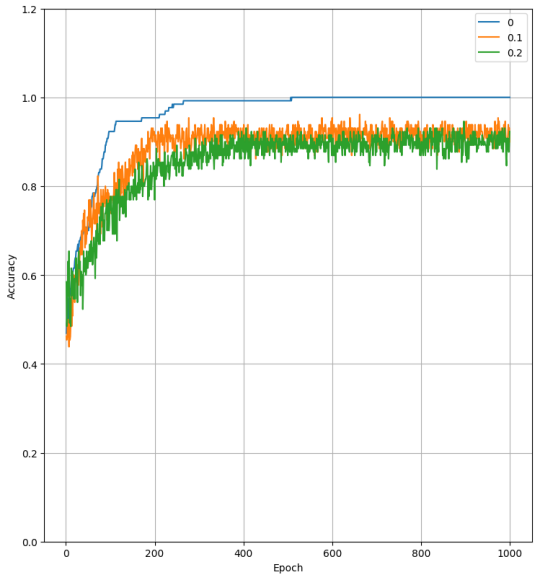

Confussion matrix
30
24
19
21
True values
Predicted values

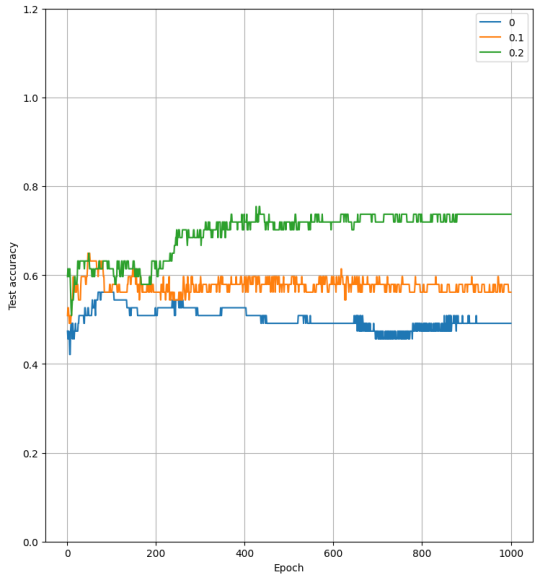
Najveće smanjenje loss-a imamo kod l2 parametra sa vrednošću 0.1, što je i očekivano. Međutim, validaciona tačnost je slična za tu situaciju, i za analizu sa vrednošću 0.001 za regularizacioni parametar, s tim što za L2 regularizaciju sa 0.001 imamo veću trening tačnost, pa ćemo u nastavku raditi sa tom vrednošću i pokušati da dodatno smanjimo preprilagođavanje.

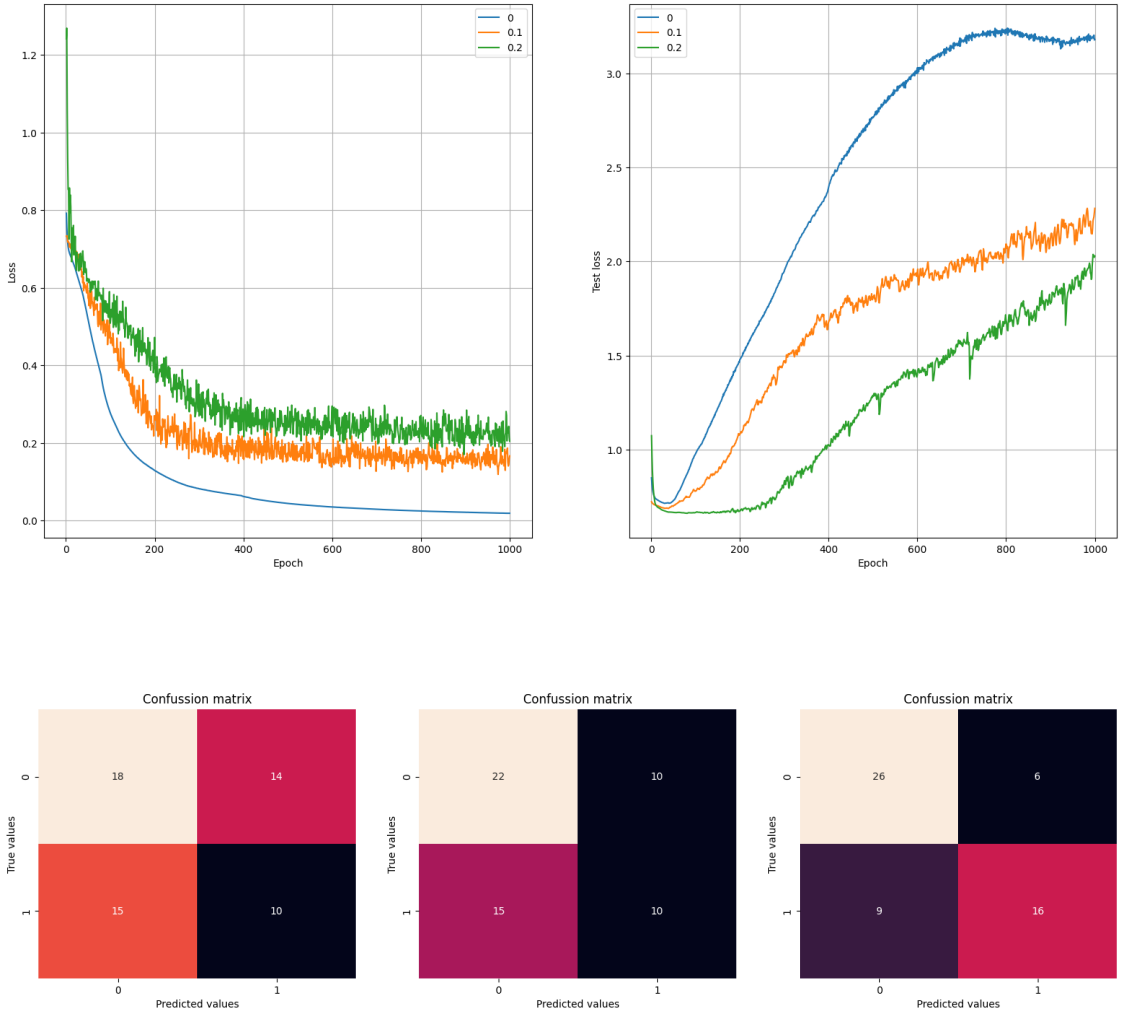
4.3.4 3.3.4. Eksperimentisanje sa dodavanjem dropout-a

[ ]:
```
dropout = [0, 0.1, 0.2]
epochs = 1000
l2 = 0.001

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0

for d in dropout:
    history, y_pred, y_val = train_model(epochs, [4, 8], lr=lr, bs=bs,␣
 ↪dropout=d, l2=l2)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=d)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=d)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=d)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()
```

```
    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
↪history['val_loss'], label=d)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```







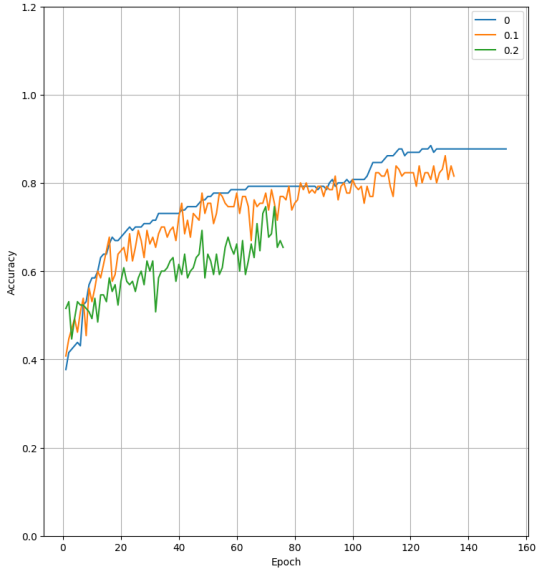
Uočavamo veći napredak prilikom dodavanja dropout-a u vrednosti od 0.2.

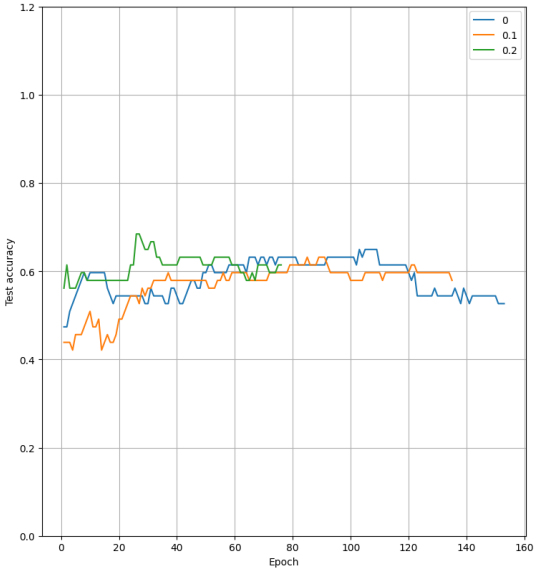
4.3.5 3.3.5. Eksperimentisanje sa ranim zaustavljanjem

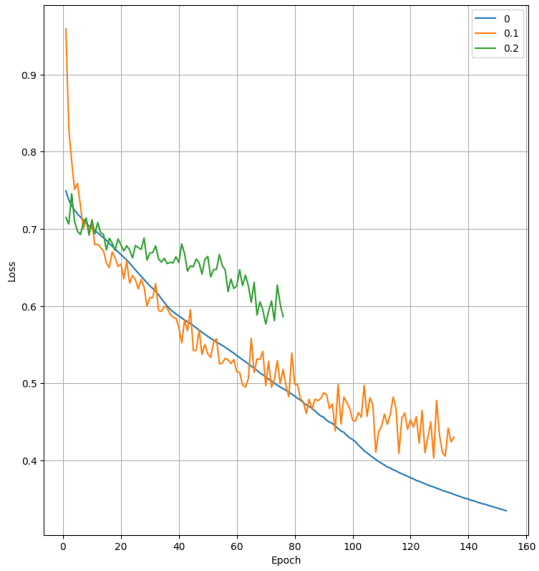
[ ]:
```
dropout = [0, 0.1, 0.2]
epochs = 1000
l2 = 0.001

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0
```

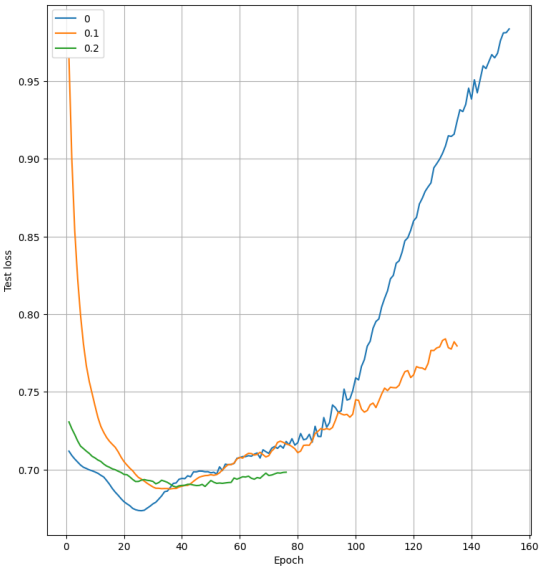
```
for d in dropout:
    history, y_pred, y_val = train_model(epochs, [4, 8], lr=lr, bs=bs,␣
 ↪dropout=d, l2=l2, early_stopping_patience=50)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=d)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=d)
    ax1[1].set_xlabel('Epoch')
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
 ↪history['loss'], label=d)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
 ↪history['val_loss'], label=d)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```

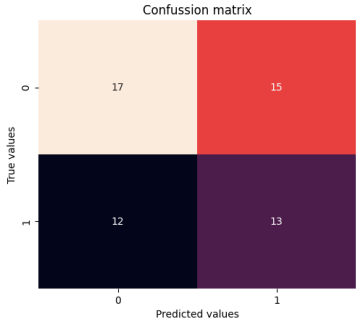
```
Epoch 153: early stopping
Epoch 135: early stopping
Epoch 76: early stopping
```

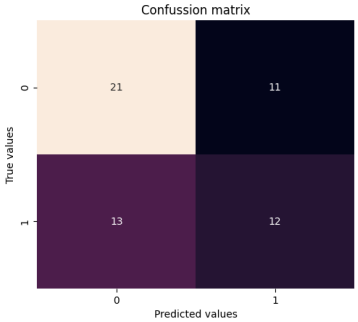

Accuracy
Epoch
0
0.1
0.2

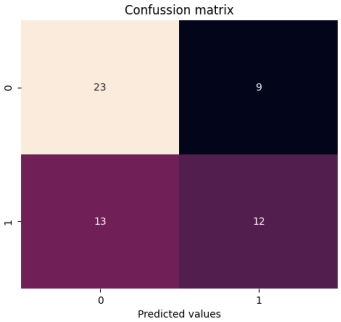

Test accuracy
Epoch
0
0.1
0.2


Loss
Epoch
0
0.1
0.2


Test loss
Epoch
0
0.1
0.2


Confussion matrix
True values
Predicted values
17
15
12
13


Confussion matrix
True values
Predicted values
21
11
13
12


Confussion matrix
True values
Predicted values
23
9
13
12

4.3.6 3.3.6. Eksperimentisanje sa augmentacijom podataka

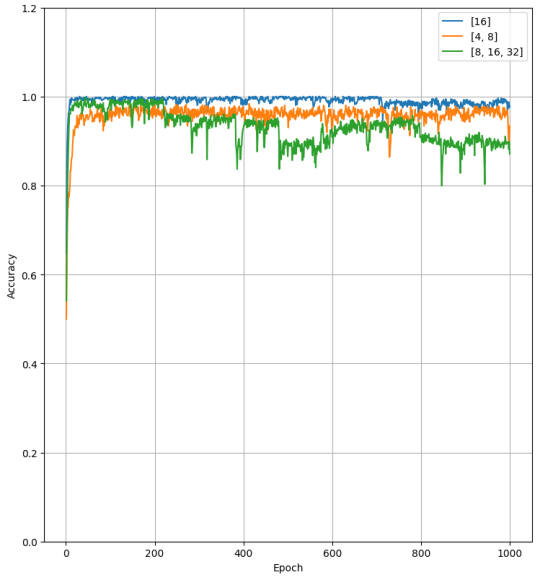
[ ]:
```
cols = ['Donorage numeric', 'Recipientage numeric', 'CD34kgx10d6 numeric',␣
 ↪'CD3dCD34 numeric', 'CD3dkgx10d8 numeric', 'Rbodymass numeric', 'ANCrecovery␣
 ↪numeric', 'PLTrecovery numeric', 'time_to_aGvHD_III_IV numeric']
```

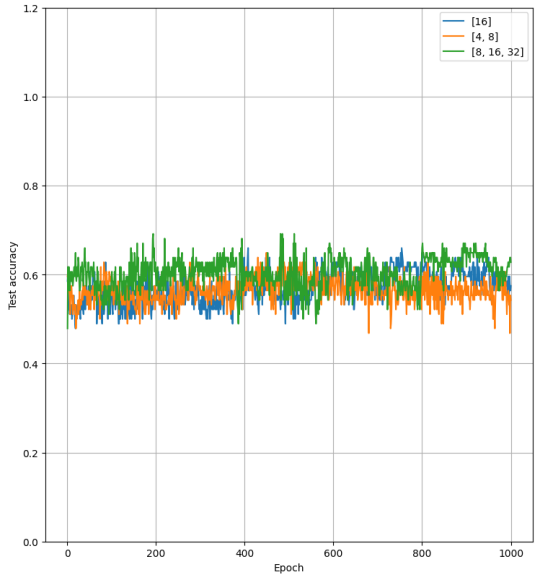
[ ]:
```
X_train_temp, X_val, y_train_temp, y_val = train_test_split(X_norm_data, y,␣
 ↪test_size=0.5, random_state=42)
X_train = np.repeat(X_train_temp, 5, axis=0)

col_idx = [X_norm.columns.get_loc(c) for c in cols if c in X_norm]
X_train[:, col_idx] = np.random.normal(0, 0.3 * np.std(X_train[:, col_idx],␣
 ↪axis=0)) + X_train[:, col_idx]
X_train = np.append(X_train_temp, X_train, axis=0)

y_train = np.append(y_train_temp, np.repeat(y_train_temp, 5, axis=0))
print(X_train.shape, y_train.shape, X_val.shape, y_val.shape)
```

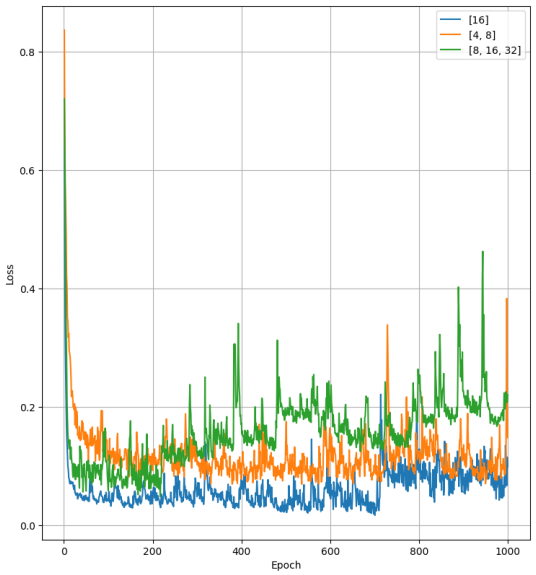
```
(558, 35) (558,) (94, 35) (94,)
```

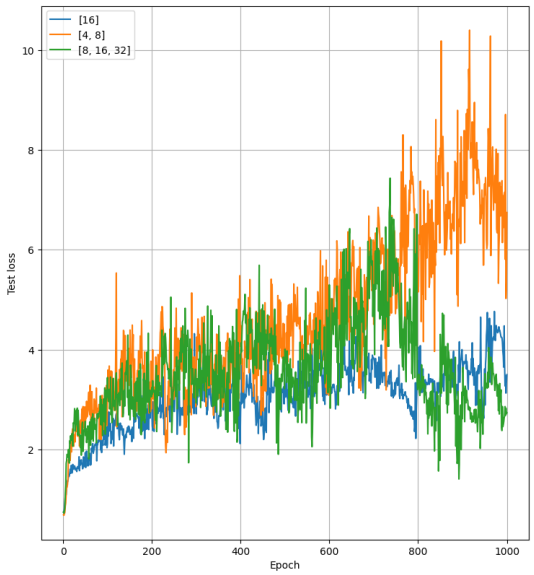
[ ]:
```
hidden_unit_size = [[16], [4, 8], [8, 16, 32]]
epochs = 1000

histories = []
fig1, ax1 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig2, ax2 = plt.subplots(1, 2, figsize=(6.4*3, 4.8*2))
fig3, ax3 = plt.subplots(1, 3, figsize=(6.4*3, 4.8))

i = 0

for hus in hidden_unit_size:
    history, y_pred, y_val = train_model(epochs, hus, lr=lr, bs=bs, l2=0.001,␣
 ↪dropout=0.1)

    ax1[0].plot(range(1, len(history.history['accuracy'])+1), history.
 ↪history['accuracy'], label=hus)
    ax1[0].set_xlabel('Epoch')
    ax1[0].set_ylabel('Accuracy')
    ax1[0].set_ylim(0, 1.2)
    ax1[0].grid()
    ax1[0].legend()

    ax1[1].plot(range(1, len(history.history['val_accuracy'])+1), history.
 ↪history['val_accuracy'], label=hus)
    ax1[1].set_xlabel('Epoch')
```

```
    ax1[1].set_ylabel('Test accuracy')
    ax1[1].set_ylim(0, 1.2)
    ax1[1].grid()
    ax1[1].legend()

    ax2[0].plot(range(1, len(history.history['loss'])+1), history.
↪history['loss'], label=hus)
    ax2[0].set_xlabel('Epoch')
    ax2[0].set_ylabel('Loss')
    ax2[0].grid()
    ax2[0].legend()

    ax2[1].plot(range(1, len(history.history['val_loss'])+1), history.
↪history['val_loss'], label=hus)
    ax2[1].set_xlabel('Epoch')
    ax2[1].set_ylabel('Test loss')
    ax2[1].grid()
    ax2[1].legend()

    cm = confusion_matrix(y_val, y_pred)
    class_labels = ['Class 0', 'Class 1']
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, ax=ax3[i])
    ax3[i].set_title("Confussion matrix")
    ax3[i].set_xlabel("Predicted values")
    ax3[i].set_ylabel("True values")
    i+=1
```









Zaključak Na osnovu svih prethodnih eksperimenata, zaključujemo da neuralna mreža nije pogodan metod za ovaj problem binarne klasifikacije. Ovo je posledica premalog skupa podataka. Najveći problem koji nastaje jeste preprilagođavanje podacima za obučavanje, što je koliko-toliko rešeno uz korišćenje ranog zaustavljanja i regularizacije. Međutim, najveća dostignuta tačnost je nešto veća od one kod linearnog klasifikatora.