SU-2019-LAB04-0036501052

January 8, 2020

Sveuilite u Zagrebu
Fakultet elektrotehnike i raunarstva

0.1 Strojno uenje 2019/2020

http://www.fer.unizg.hr/predmet/su

0.1.1 Laboratorijska vjeba 4: Ansambli i procjena parametara

Verzija: 0.4
Zadnji put aurirano: 27. rujna 2019.

Objavljeno: **30. rujna 2019.**
Rok za predaju: **16. prosinca 2019. u 07:00h**

0.1.2 Upute

etvrta laboratorijska vjeba sastoji se od **etiri** zadatka. Kako bi kvalitetnije, ali i na manje zamoran nain usvojili gradivo ovog kolegija, potrudili smo se ukljuiti tri vrste zadataka: **1)** implementacija manjih algoritama, modela ili postupaka; **2)** eksperimenti s raznim modelima te njihovim hiperparametrima, te **3)** primjena modela na (stvarnim) podatcima. Ovim zadatcima pokrivamo dvije paradigme uenja: uenje izgradnjom (engl. *learning by building*) i uenje eksperimentiranjem (engl. *learning by experimenting*).

U nastavku slijedite upute navedene u elijama s tekstom. Rjeavanje vjebe svodi se na **dopunjavanje ove biljenice**: umetanja elije ili vie njih **ispod** teksta zadatka, pisanja odgovarajueg kôda te evaluiranja elija.

Osigurajte da u potpunosti **razumijete** kôd koji ste napisali. Kod predaje vjebe, morate biti u stanju na zahtjev asistenta (ili demonstratora) preinaiti i ponovno evaluirati Va kôd. Nadalje, morate razumjeti teorijske osnove onoga to radite, u okvirima onoga to smo obradili na predavanju. Ispod nekih zadataka moete nai i pitanja koja slue kao smjernice za bolje razumijevanje gradiva (**nemojte pisati** odgovore na pitanja u biljenicu). Stoga se nemojte ograniiti samo na to da rijeite zadatak, nego slobodno eksperimentirajte. To upravo i jest svrha ovih vjebi.

Vjebe trebate raditi **samostalno**. Moete se konzultirati s drugima o naelnom nainu rjeavanja, ali u konanici morate sami odraditi vjebu. U protivnome vjeba nema smisla.

[2]:
```
# Uitaj osnovne biblioteke...
import sklearn
import mlutils
import numpy as np
import matplotlib.pyplot as plt
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

0.1.3 1. Ansambli (glasovanje)

(a) Va je zadatak napisati razred `VotingClassifierDIY` koji implementira glasaki ansambl. Konstruktor razreda ima **dva** parametra: `clfs` koji predstavlja listu klasifikatora (objekata iz paketa `sklearn`) i `voting_scheme` koji oznaava radi li se o glasovanju prebrojavanjem (`SCHEME_COUNTING`) ili usrednjavanjem (`SCHEME_AVERAGING`). Glasovanje prebrojavanjem jednostavno vraa najeu oznaku klase, dok glasovanje usrednjavanjem uprosjeuje pouzdanosti klasifikacije u neku klasu (po svim klasifikatorima) te vraa onu s najveom pouzdanou. Primijetite da svi klasifikatori imaju jednake teine. O komplementarnosti klasifikatora vodimo rauna tako da koristimo jednake klasifikatore s razliitim hiperparametrima.

Razred sadrava metode `fit(X, y)` za uenje ansambla i dvije metode za predikciju: `predict(X)` i `predict_proba(X)`. Prva vraa predviene oznake klasa, a druga vjerojatnosti pripadanja svakoj od klasa za svaki od danih primjera iz `X`.

NB: Jedan od razreda koji bi Vam mogao biti koristan jest `collections.Counter`. Takoer vrijedi i za funkcije `numpy.argmax` i `numpy.dstack`.

[239]:
```
from collections import Counter

class VotingClassifierDIY(object):

    SCHEME_COUNTING = "counting"
    SCHEME_AVERAGING = "averaging"

    def __init__(self, clfs, voting_scheme=SCHEME_COUNTING):
        self.clfs = clfs
        self.voting_scheme = voting_scheme

    def fit(self, X, y):
        for clf in self.clfs:
            clf.fit(X,y)

    def predict_proba(self, X):
        if self.voting_scheme == self.SCHEME_COUNTING:
            raise RuntimeError(f"predict_proba is not available when voting is␣
 ↪{self.voting_scheme}")
        probs = []
        probs = self.clfs[0].predict_proba(X)
        for clf in self.clfs[1:]:
            p = clf.predict_proba(X)
```

```
            probs = np.dstack((probs, p))
        return probs.mean(axis=2)

    def predict(self, X):
        preds = np.array([])
        for clf in clfs:
            y_pred = clf.predict(X)
            preds = np.concatenate((preds, y_pred))
        return preds
```

(b) Uvjerite se da Vaa implementacija radi jednako onoj u razredu `ensemble.VotingClassifier`, i to pri oba naina glasovanja (parametar `voting`). Parametar `weights` ostavite na pretpostavljenoj vrijednosti. Za ovu provjeru koristite tri klasifikatora logistike regresije s razliitom stopom regularizacije i brojem iteracija. Koristite skup podataka dan u nastavku. Ekvivalentnost implementacije najlake je provjeriti usporedbom izlaza funkcije `predict` (kod prebrojavanja) i funkcije `predict_proba` (kod usrednjavanja).

NB: Ne koristimo SVM jer njegova ugraena (probabilistika) implementacija nije posve deterministika, to bi onemoguilo robusnu provjeru Vae implementacije.

[164]:
```
from sklearn.datasets import make_classification
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression

X_voting, y_voting = make_classification(n_samples=1000, n_features=4,␣
 ↪n_redundant=0, n_informative=3, n_classes=3, n_clusters_per_class=2)
print(X_voting.shape)

clf1 = LogisticRegression(solver="newton-cg", max_iter=100, C = 0.8)
clf2 = LogisticRegression(solver="newton-cg", max_iter=500, C = 0.5)
clf3 = LogisticRegression(solver="newton-cg", max_iter=1000, C = 0.1)
```

```
(1000, 4)
```

[248]:
```
import warnings
warnings.filterwarnings("ignore")
X = X_voting
y = y_voting

clfs  = [clf1,clf2,clf3]
m = VotingClassifierDIY(clfs,"averging")
m.fit(X,y)
probs = m.predict_proba(X)
print(probs)


print("\n-----------------------------------\n")
```

```
model = VotingClassifier(estimators=[('lr1', clf1),
                                     ('lr2', clf2),
                                     ('lr3', clf3)], voting="soft")
model.fit(X,y)
pr = model.predict_proba(X)
print(pr)
```

```
[[0.68885555 0.26143339 0.04971106]
 [0.67849018 0.24269535 0.07881448]
 [0.35568403 0.40035245 0.24396353]
 ...
 [0.21567201 0.43838381 0.34594418]
 [0.57076317 0.29170995 0.13752688]
 [0.26966207 0.43610395 0.29423399]]

----------------------------------

[[0.68885555 0.26143339 0.04971106]
 [0.67849018 0.24269535 0.07881448]
 [0.35568403 0.40035245 0.24396353]
 ...
 [0.21567201 0.43838381 0.34594418]
 [0.57076317 0.29170995 0.13752688]
 [0.26966207 0.43610395 0.29423399]]
```

Q: Kada je prebrojavanje bolje od usrednjavanja? Zato? A obratno?
Q: Bi li se ovakav algoritam mogao primijeniti na regresiju? Kako?

0.1.4 2. Ansambli (*bagging*)

U ovom zadatku ete isprobati tipinog predstavnika *bagging*-algoritma, **algoritam sluajnih uma**. Pitanje na koje elimo odgovoriti jest kako se ovakvi algoritmi nose s prenauenou, odnosno, smanjuje li *bagging* varijancu modela.

Eksperiment ete provesti na danom skupu podataka:

[321]:
```
from sklearn.model_selection import train_test_split

X_bag, y_bag = make_classification(n_samples=1000, n_features=20,␣
 ↪n_redundant=1, n_informative=17, n_classes=3, n_clusters_per_class=2)
X_bag_train, X_bag_test, y_bag_train, y_bag_test = train_test_split(X_bag,␣
 ↪y_bag, train_size=0.7, random_state=69)

print(X_bag.shape)
```

```
(1000, 20)
```

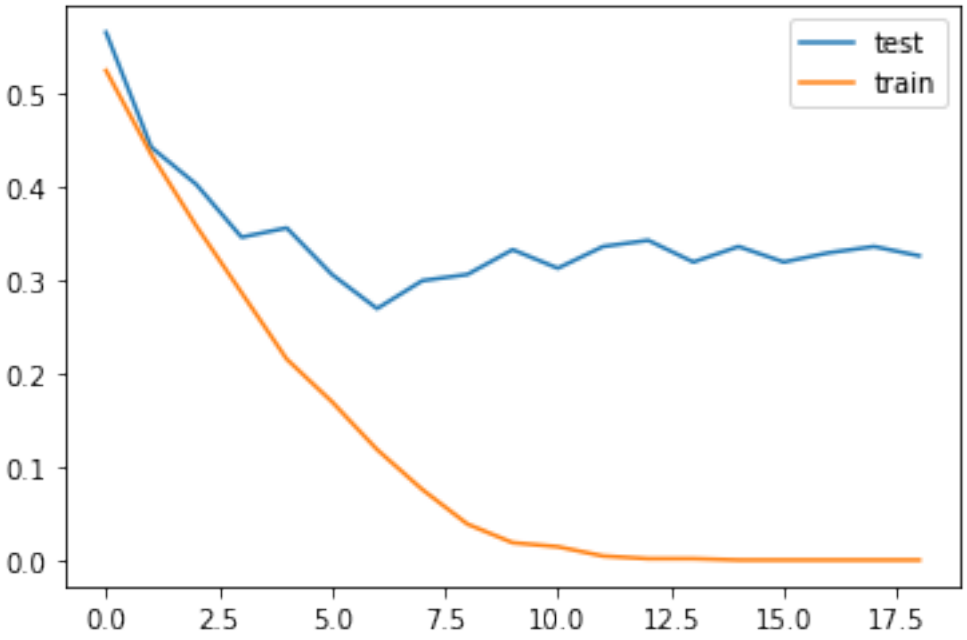
Razred koji implementira stablo odluke jest `tree.DecisionTreeClassifier`. Prvo nauite **stablo odluke** (engl. *decision tree*) na skupu za uenje, ali tako da je taj model presloen. To moete postii

tako da poveate najveu moguu dubinu stabla (parametar `max_depth`). Ispiite pogreku na skupu za ispitivanje (pogreku 0-1; pogledajte paket `metrics`).

[356]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import zero_one_loss
import pdb

def get_errors(model_init, X_test, y_test, X_train, y_train,N=20):
    E_test  = []
    E_train = []
    for depth in range(1,N):
        model  = model_init(depth)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        e = zero_one_loss(y_pred, y_test)
        E_test.append(e)

        y_pred = model.predict(X_train)
        e = zero_one_loss(y_pred, y_train)
        E_train.append(e)
    return E_train, E_test
```

[357]:
```
E_train, E_test = get_errors(lambda depth : DecisionTreeClassifier(max_depth =␣
 ↪depth), X_bag_test, y_bag_test, X_bag_train, y_bag_train)

plt.plot(E_test, label="test")
plt.plot(E_train, label="train")
plt.legend()
```

[357]:
```
<matplotlib.legend.Legend at 0x7f47692961d0>
```

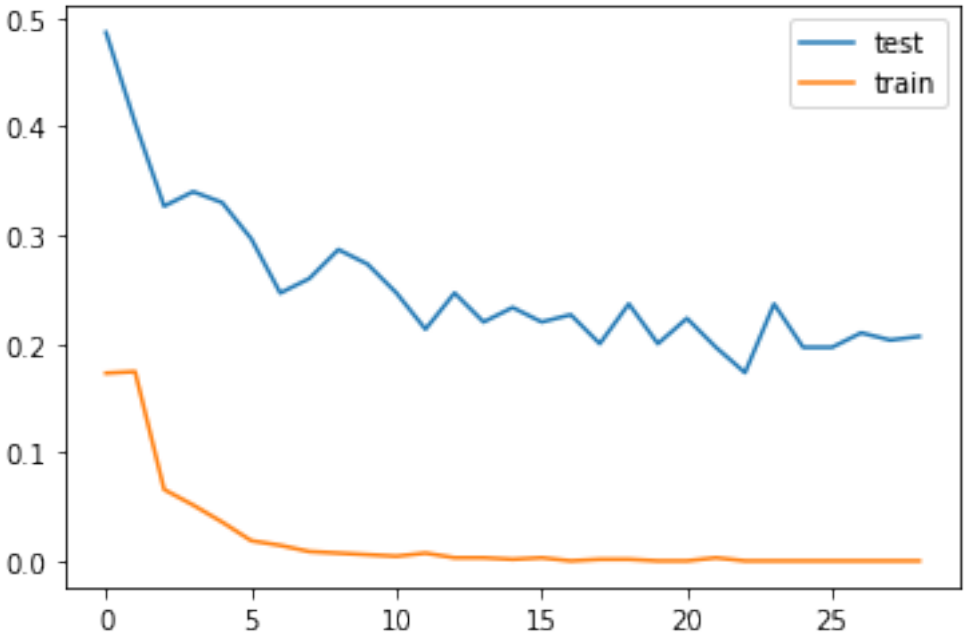


Sada isprobajte algoritam sluajnih uma (dostupan u razredu `ensemble.RandomForestClassifier`) za razliit broj stabala $L \in [1,30]$. Iscrtajte pogreku na skupu za uenje i na skupu za ispitivanje u ovisnosti o tom hiperparametru. Ispiite najmanju pogreku na skupu za ispitivanje.

[274]:
```
from sklearn.ensemble import RandomForestClassifier
```

[359]:
```
E_train, E_test = get_errors(lambda i : RandomForestClassifier(n_estimators=i),␣
↪X_bag_test, y_bag_test, X_bag_train, y_bag_train,N=30)

plt.plot(E_test, label="test")
plt.plot(E_train, label="train")
plt.legend()
```

[359]:
```
<matplotlib.legend.Legend at 0x7f47690f9f98>
```



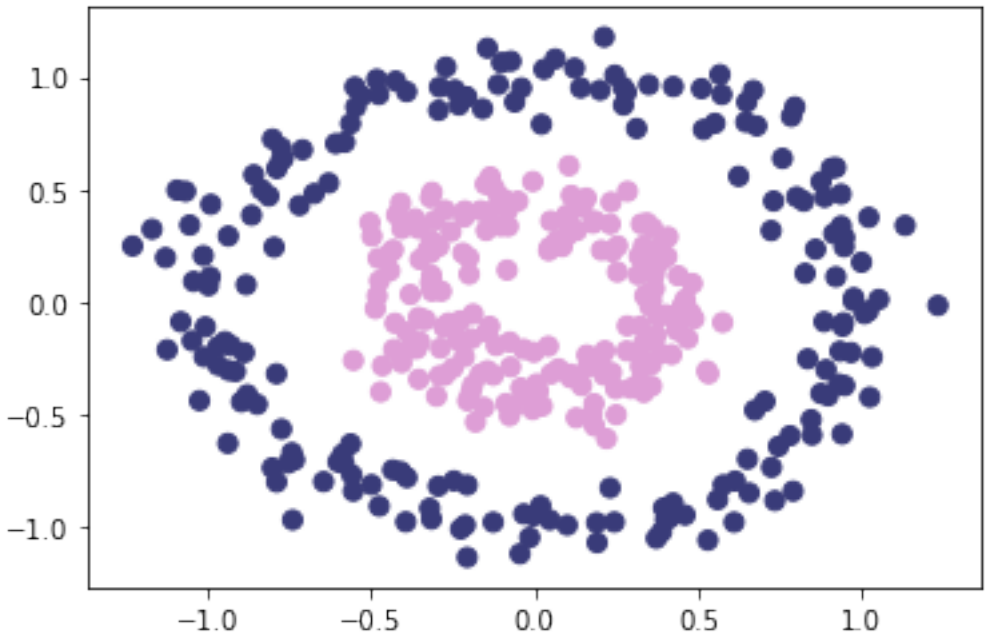
Q: to moete zakljuiti iz ovih grafikona?
Q: Kako *bagging* postie diverzifikaciju pojedinanih osnovnih modela?
Q: Koristi li ovaj algoritam sloeni ili jednostavni osnovni model? Zato?

0.1.5 3. Ansambli (*boosting*)

U ovom zadatku pogledat emo klasifikacijski algoritam AdaBoost, koji je implementiran u razredu `ensemble.AdaBoostClassifier`. Ovaj algoritam tipian je predstavnik *boosting*-algoritama.

Najprije emo generirati eksperimentalni skup podataka koristei `datasets.make_circles`. Ova funkcija stvara dvodimenzijski klasifikacijski problem u kojem su dva razreda podataka rasporeena u obliku krunica, tako da je jedan razred unutar drugog.

[354]:
```
from sklearn.datasets import make_circles

circ_X, circ_y = make_circles(n_samples=400, noise=0.1, factor=0.4)
mlutils.plot_2d_clf_problem(circ_X, circ_y)
```



(a) *Boosting*, kao vrsta ansambla, takoer se temelji na kombinaciji vie klasifikatora s ciljem boljih prediktivnih sposobnosti. Meutim, ono to ovakav tip ansambla ini zanimljivim jest to da za osnovni klasifikator trai **slabi klasifikator** (engl. *weak classifier*), odnosno klasifikator koji radi tek malo bolje od nasuminog pogaanja. esto koriteni klasifikator za tu svrhu jest **panj odluke** (engl. *decision stump*), koji radi predikciju na temelju samo jedne znaajke ulaznih primjera. Panj odluke specijalizacija je **stabla odluke** (engl. *decision tree*) koje smo ve spomenuli. Panj odluke stablo je dubine 1. Stabla odluke implementirana su u razredu `tree.DecisionTreeClassifier`.

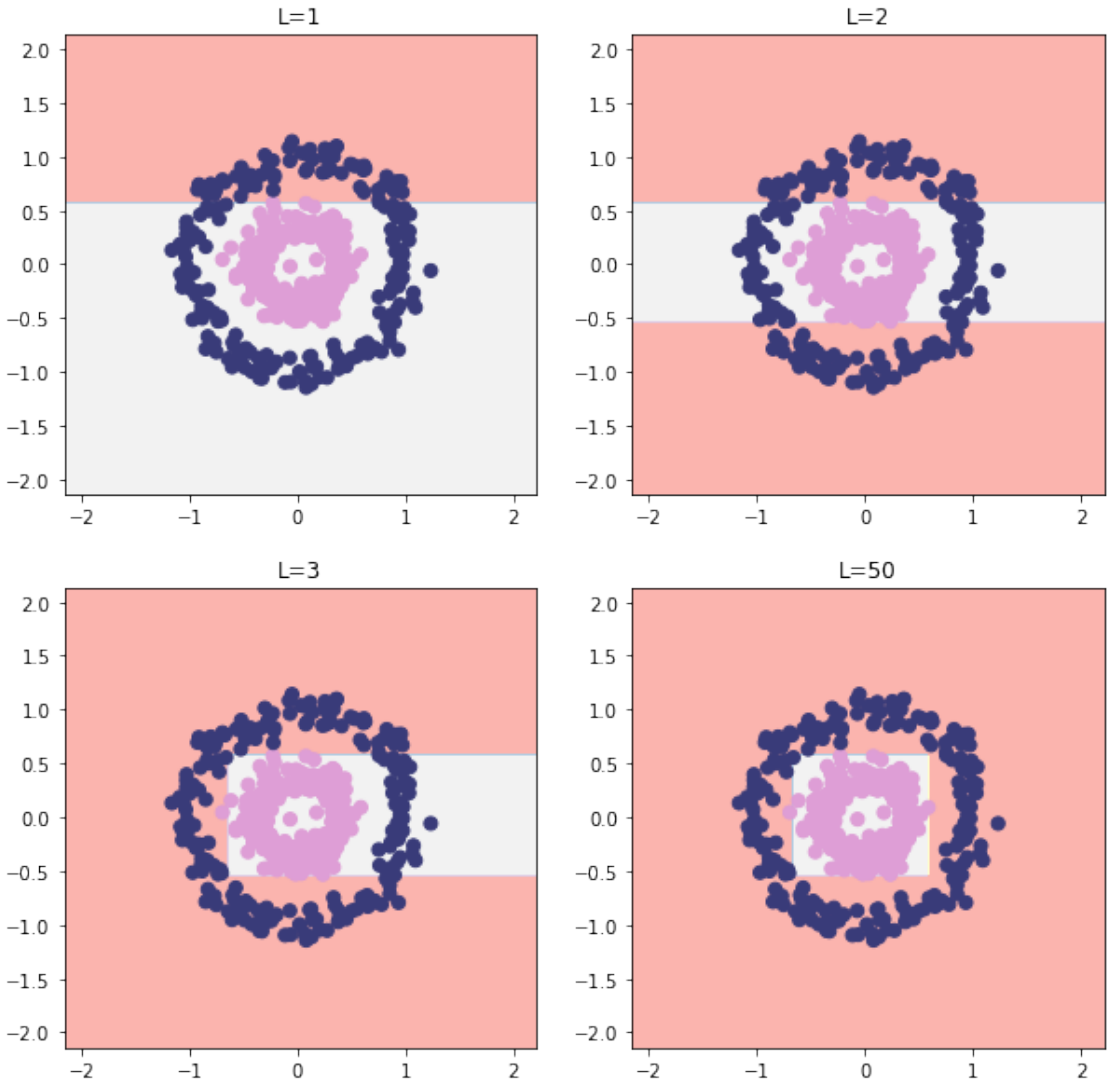
Radi ilustracije, nauite ansambl (AdaBoost) koristei panj odluke kao osnovni klasifikator, ali pritom isprobavajui razliit broj klasifikatora u ansamblu iz skupa $L \in \{1, 2, 3, 50\}$. Prikaite decizijske granice na danom skupu podataka za svaku od vrijednosti koritenjem pomone funkcije `mlutils.plot_2d_clf_problem`.

NB: Jo jedan dokaz da hrvatska terminologija zaista moe biti smijena. :)

[355]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

X = circ_X
y = circ_y
```

[302]:
```
Ls = [1,2,3,50]
```

```
plt.figure(figsize=(10,10))
for i,L in enumerate(Ls):
    base = get_trees(3)
    model = AdaBoostClassifier(n_estimators=L)
    model.fit(X,y)
    plt.subplot(2,2,i+1)
    plt.title(f"L={L}")
    mlutils.plot_2d_clf_problem(X, y, h = model.predict)
```



Q: Kako AdaBoost radi? Ovise li izlazi pojedinih osnovnih modela o onima drugih?
Q: Je li AdaBoost linearan klasifikator? Pojasnite.

(b) Kao to je i za oekivati, broj klasifikatora L u ansamblu predstavlja hiperparametar algoritma *AdaBoost*. U ovom zadatku prouit ete kako on utjee na generalizacijsku sposobnost Vaeg ansambla.

Ponovno, koristite panj odluke kao osnovni klasifikator.

Posluite se skupom podataka koji je dan nie.

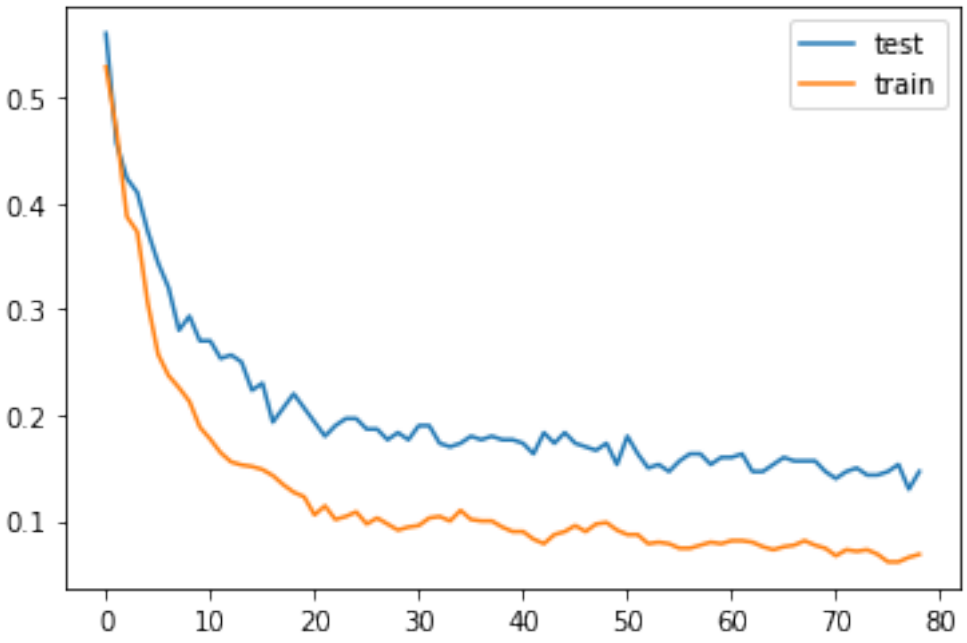
[305]:
```
from sklearn.model_selection import train_test_split

X_boost, y_boost = make_classification(n_samples=1000, n_features=20,␣
 ↪n_redundant=0, n_informative=18, n_classes=3, n_clusters_per_class=1)
X_boost_train, X_boost_test, y_boost_train, y_boost_test =␣
 ↪train_test_split(X_boost, y_boost, train_size=0.7, random_state=69)
X_boost.shape
```

[305]:
```
(1000, 20)
```

Iscrtajte krivulje pogreaka na skupu za uenje i ispitivanje u ovisnosti o hiperparametru $L \in [1, 80]$. Koristite pogreku 0-1 iz paketa `metrics`. Ispiite najmanju ostvarenu pogreku na skupu za ispitivanje, te pripadajuu vrijednost hiperparametra L.

[367]:
```
E_train, E_test = get_errors(lambda i : AdaBoostClassifier(n_estimators=i),␣
 ↪X_boost_test, y_boost_test, X_boost_train, y_boost_train, N=80)

plt.plot(E_test, label="test")
plt.plot(E_train, label="train")
plt.legend()
```

[367]:
```
<matplotlib.legend.Legend at 0x7f4768ef07f0>
```



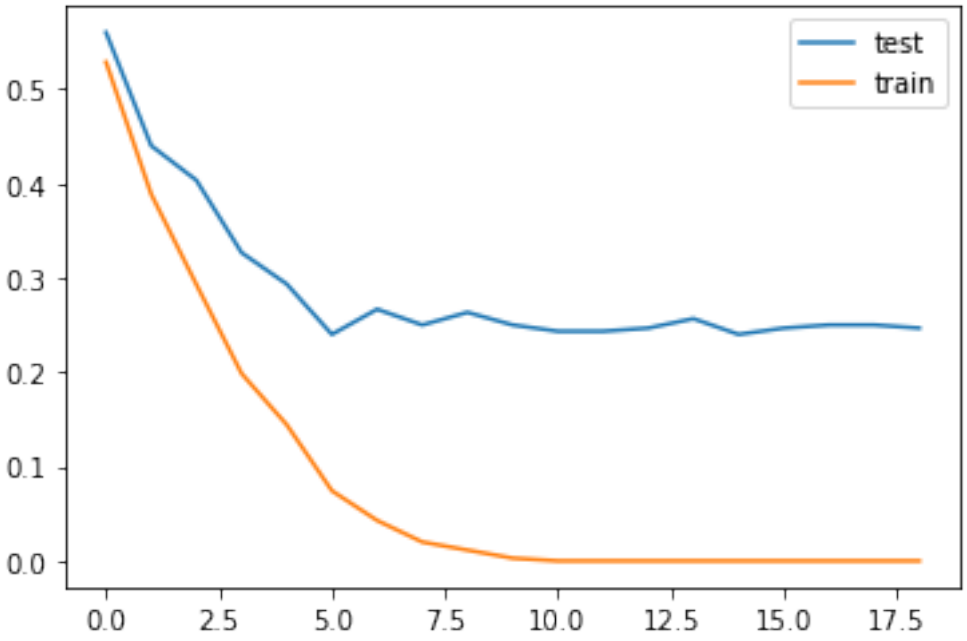
Q: Moe li uope doi do prenauenosti pri koritenju *boosting*-algoritama?

(c) Kao to je reeno na poetku, *boosting*-algoritmi trae slabe klasifikatore kako bi bili najefikasniji to mogu biti. Meutim, kako se takav ansambl mjeri s jednim **jakim klasifikatorom** (engl. *strong classifier*)? To emo isprobati na istom primjeru, ali koritenjem jednog optimalno nauenog stabla odluke.

Ispiite pogreku ispitivanja optimalnog stabla odluke. Glavni hiperparametar stabala odluka jest njihova maksimalna dubina d (parametar `max_depth`). Iscrtajte krivulje pogreaka na skupu za uenje i ispitivanje u ovisnosti o dubini stabla $d \in [1, 20]$.

[380]:
```
E_train, E_test = get_errors(lambda i : DecisionTreeClassifier(max_depth=i),␣
↪X_boost_test, y_boost_test, X_boost_train, y_boost_train, N=20)

plt.plot(E_test, label="test")
plt.plot(E_train, label="train")
plt.legend()

print(f"optimal_depth={np.argmin(E_test)}")
```

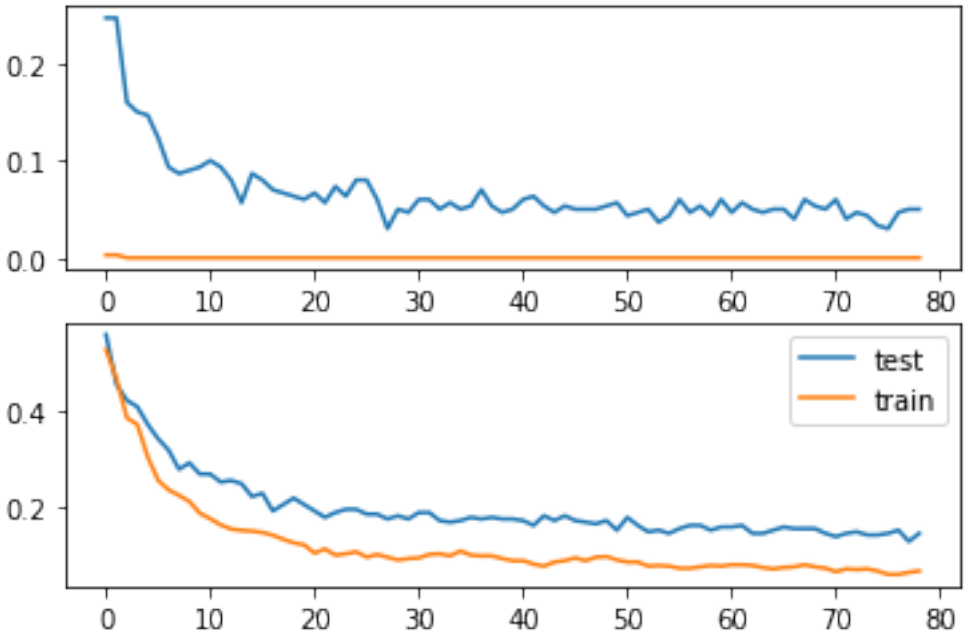
```
optimal_depth=5
```



[390]:
```
E_train1, E_test1 = get_errors(lambda i :␣
↪AdaBoostClassifier(DecisionTreeClassifier(max_depth=10),algorithm="SAMME",␣
↪n_estimators=i), X_boost_test, y_boost_test, X_boost_train, y_boost_train,␣
↪N=80)
E_train2, E_test2 = get_errors(lambda i : AdaBoostClassifier(n_estimators=i),␣
↪X_boost_test, y_boost_test, X_boost_train, y_boost_train, N=80)
```

```
plt.subplot(2,1,1)
plt.plot(E_test1, label="test")
plt.plot(E_train1, label="train")

plt.subplot(2,1,2)
plt.plot(E_test2, label="test")
plt.plot(E_train2, label="train")

plt.legend()
```

[390]: `<matplotlib.legend.Legend at 0x7f4763aa5fd0>`



Q: Isplati li se koristiti ansambl u obliku *boostinga*? Idu li grafikoni tome u prilog? **Q:** Koja je prednost *boostinga* nad koritenjem jednog jakog klasifikatora?

0.1.6 4. Procjena maksimalne izglednosti i procjena maksimalne aposteriorne vjerojatnosti

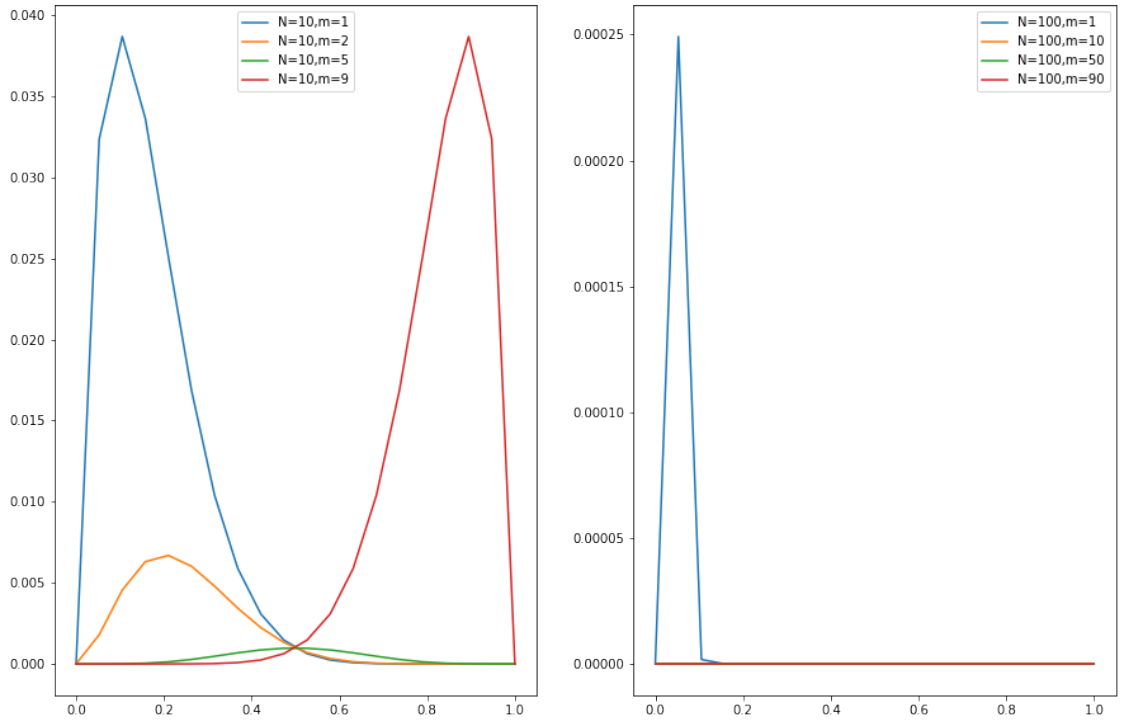
(a) Definirajte funkciju izglednosti $\mathcal{L}(\mu|\mathcal{D})$ za skup $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$ Bernoullijevih varijabli. Neka od N varijabli njih m ima vrijednost 1 (npr. od N bacanja novia, m puta smo dobili glavu). Definirajte funkciju izglednosti tako da je parametrizirana s N i m, dakle definirajte funkciju $\mathcal{L}(\mu|N,m)$.

[393]:
```
def bernoulli_likelihood(mu, N, m):
    """
        N : trials
        m : experiment successes
    """
    return (mu**m)*(1-mu)**(N-m)
```

Prikaite funkciju $\mathcal{L}(\mu|N,m)$ za (1) $N = 10$ i $m = 1,2,5,9$ te za (2) $N = 100$ i $m = 1,10,50,90$ (dva zasebna grafikona).

[424]:
```
N  = 10
ms = [1,2,5,9]

plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
for i,m in enumerate(ms):
    x = np.linspace(0,1,20)
    plt.plot(x,bernoulli_likelihood(x,N,m),label=f"N={N},m={m}")
plt.legend()

N  = 100
ms = [1,10,50,90]
plt.subplot(1,2,2)
for i,m in enumerate(ms):
    x = np.linspace(0,1,20)
    plt.plot(x,bernoulli_likelihood(x,N,m),label=f"N={N},m={m}")
plt.legend()
```

[424]:
```
<matplotlib.legend.Legend at 0x7f4768539f98>
```

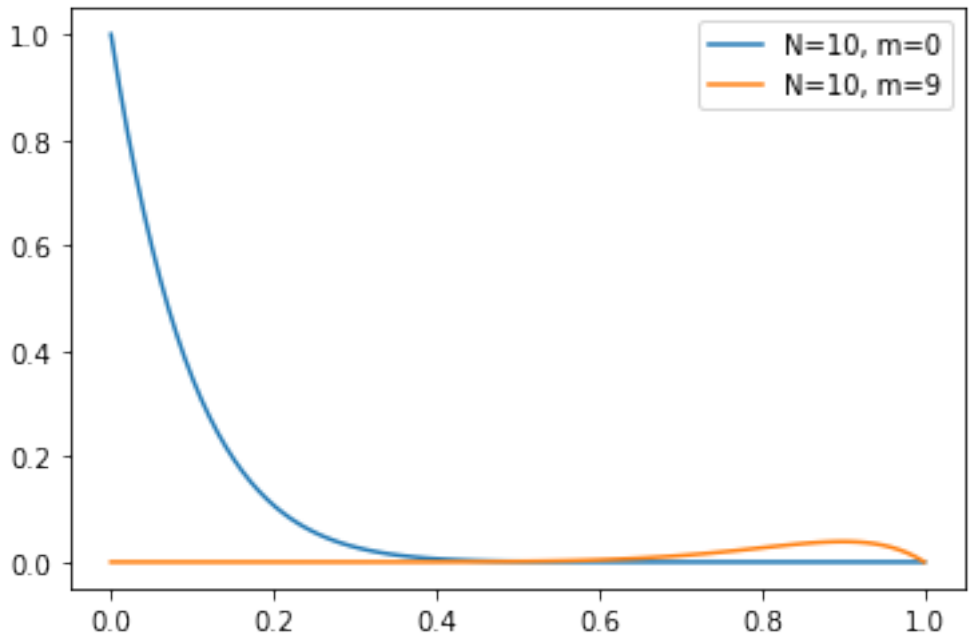


Q: Koja vrijednost odgovara ML-procjenama i zato?

(c) Prikaite funkciju $\mathcal{L}(\mu|N,m)$ za $N = 10$ i $m = \{0,9\}$.

[429]:
```
x = np.linspace(0,1,100)
plt.plot(x,bernoulli_likelihood(x, N = 10, m = 0), label = "N=10, m=0")
plt.plot(x,bernoulli_likelihood(x, N = 10, m = 9), label = "N=10, m=9")
plt.legend()
```

[429]:
```
<matplotlib.legend.Legend at 0x7f47682f3da0>
```



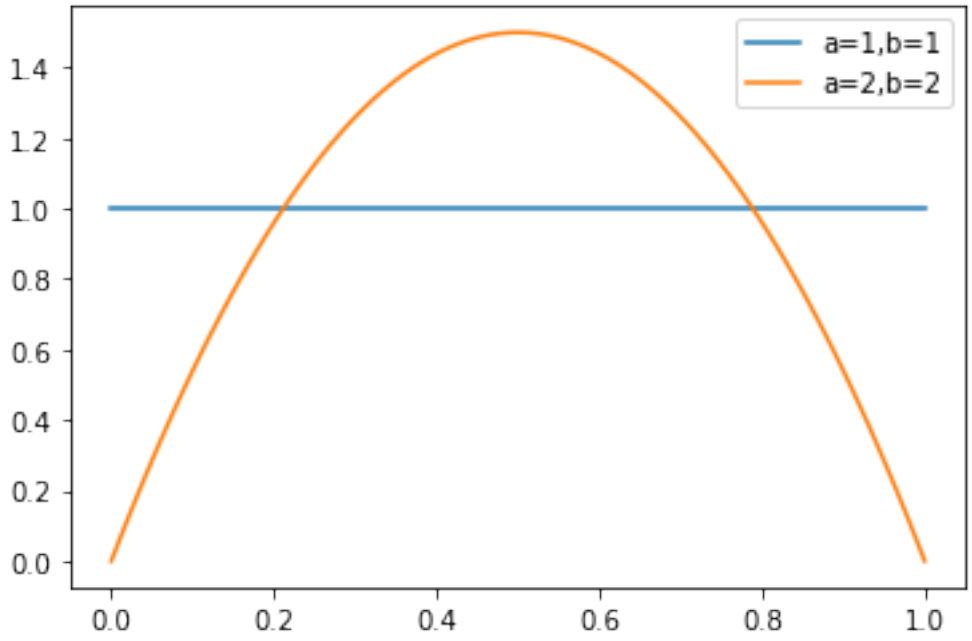
Q: Koja je ML-procjena za μ i to je problem s takvom procjenom u ovome sluaju?

(d) Prikaite beta-distribuciju $B(\mu|\alpha,\beta)$ za razliite kombinacije parametara α i β, ukljuivo $\alpha = \beta = 1$ te $\alpha = \beta = 2$.

[445]:
```
from scipy.stats import beta
```

[453]:
```
x = np.linspace(0,1,100)
plt.plot(x,beta.pdf(x,a=1,b=1),label="a=1,b=1")
plt.plot(x,beta.pdf(x,a=2,b=2),label="a=2,b=2")
plt.legend()
```

[453]:
```
<matplotlib.legend.Legend at 0x7f476829d7f0>
```



[491]:
```
beta.pdf(0.5,a=2,b=2)
```

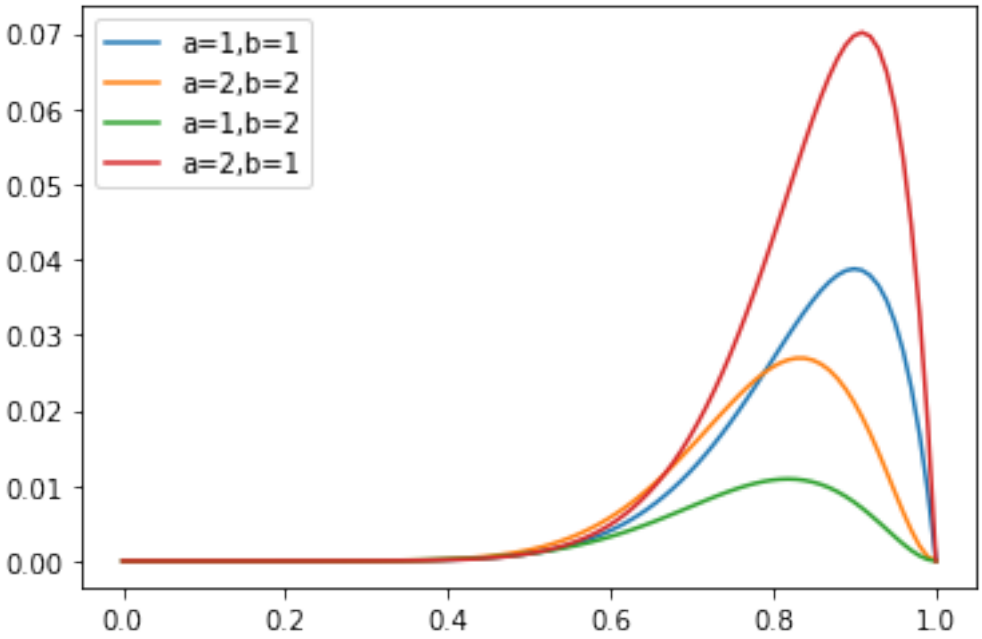
[491]:
```
1.5
```

Q: Koje parametere biste odabrali za modeliranje apriornog znanja o parametru μ za novi za koji mislite da je "donekle pravedan, ali malo ee pada na glavu"? Koje biste parametre odabrali za novi za koji drite da je posve pravedan? Zato uope koristimo beta-distribuciju, a ne neku drugu?

(e) Definirajte funkciju za izraun zajednike vjerojatnosti $P(\mu, \mathcal{D}) = P(\mathcal{D}|\mu) \cdot P(\mu|\alpha, \beta)$ te prikaite tu funkciju za $N = 10$ i $m = 9$ i nekolicinu kombinacija parametara α i β.

[473]:
```
def posterior(mu, N, m, a, b):
    likelihood = bernoulli_likelihood(mu,N,m)
    prior      = beta.pdf(mu,a=a,b=b)
    return likelihood*prior



plt.plot(x,posterior(x, N=10, m=9, a=1, b=1), label="a=1,b=1")
plt.plot(x,posterior(x, N=10, m=9, a=2, b=2), label="a=2,b=2")
plt.plot(x,posterior(x, N=10, m=9, a=1, b=2), label="a=1,b=2")
plt.plot(x,posterior(x, N=10, m=9, a=2, b=1), label="a=2,b=1")
plt.legend()
```

[473]:
```
<matplotlib.legend.Legend at 0x7f4762409c88>
```

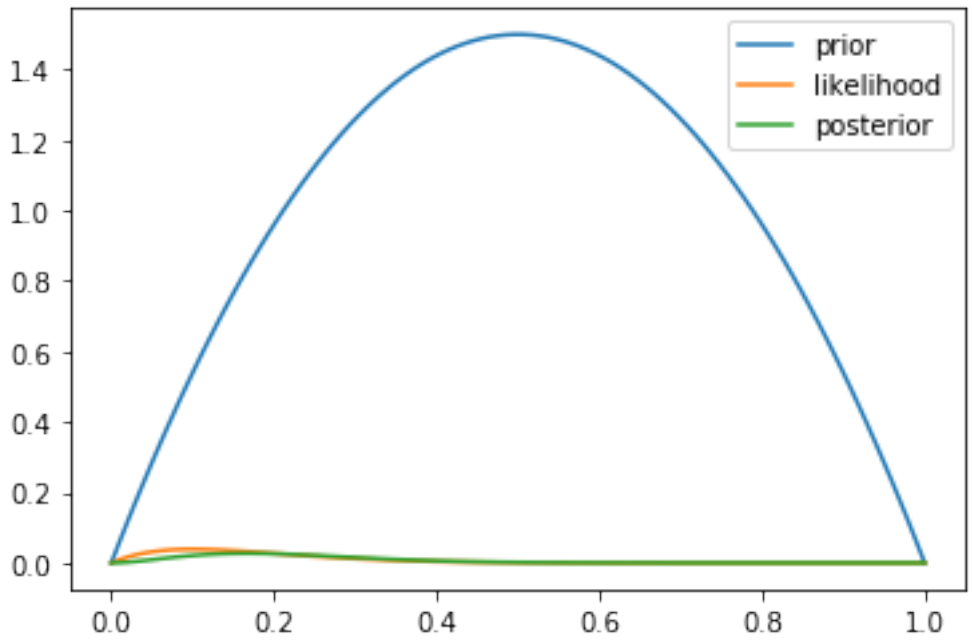


Q: Koje vrijednosti odgovaraju MAP-procjeni za μ? Usporedite ih sa ML-procjenama.

(f) Za $N = 10$ i $m = 1$, na jednome grafikonu prikaite sve tri distribucije: $P(\mu, \mathcal{D})$, $P(\mu|\alpha, \beta)$ i $\mathcal{L}(\mu|\mathcal{D})$.

[493]:
```
a = 2
b = 2
prior      = beta.pdf(x, a=a, b=b)
likelihood = bernoulli_likelihood(x, N=10, m=1)
post       = posterior(x, N=10, m=1, a=a, b=b)

plt.plot(x,prior, label="prior")
plt.plot(x,likelihood, label="likelihood")
plt.plot(x,post, label="posterior")

plt.legend()
```

[493]:
```
<matplotlib.legend.Legend at 0x7f4762cf9fd0>
```

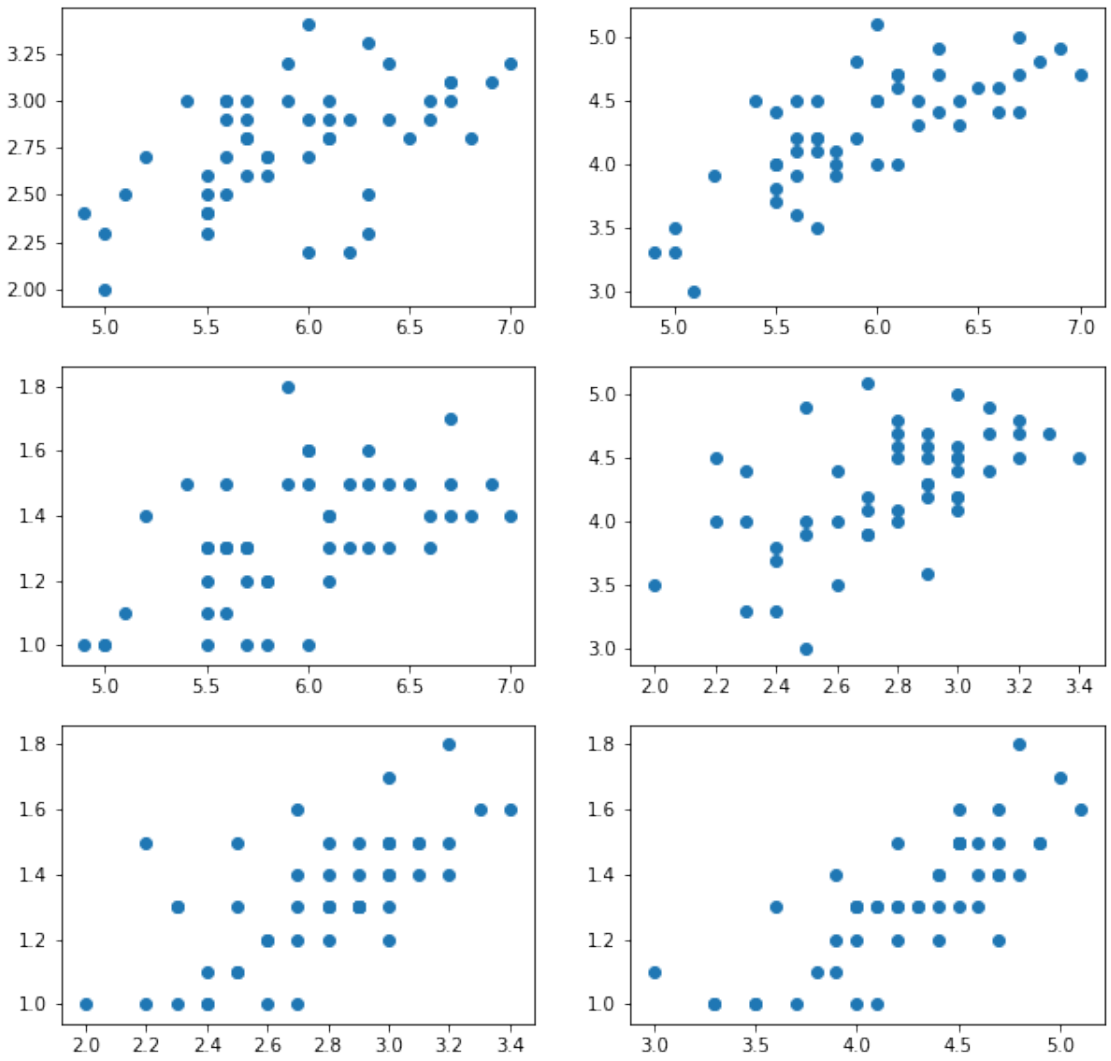


(g) Proitajte ove upute o uitavanju oglednih skupova podataka. Uitajte skup podataka *Iris*. Taj skup sadri $n = 4$ znaajke i $K = 3$ klase. Odaberite jednu klasu i odaberite sve primjere iz te klase, dok ostale primjere zanemarite (**u nastavku radite iskljuivo s primjerima iz te jedne klase**). Vizualizirajte podatke tako da nainite 2D-prikaze za svaki par znaajki (est grafikona; za prikaz je najjednostavnije koristiti funkciju `scatter`).

NB: Mogla bi Vam dobro dui funkcija `itertools.combinations`.

[494]:
```
from sklearn.datasets import load_iris
import itertools as it
```

[520]:
```
X,y = load_iris(True)
X = X[y == 1]
y = y[y == 1]


xs = [0,1,2,3]
plt.figure(figsize=(10,10))
for ix,c in enumerate(it.combinations(xs,2)):
    i,j = c
    plt.subplot(3,2,ix+1)
    plt.scatter(X[:,i], X[:,j])
```



(h) Pogledajte opis modul `stats` te prouite funkciju `norm`. Implementirajte funkciju log-izglednosti za parametre μ i σ^2 normalne distribucije.

[543]:
```
from scipy.stats import norm
```

[587]:
```
def normal_loglikelihood(x, mu, var):
    probs = norm.pdf(x,loc=mu,scale=var)
    prod  = np.product(probs)
    return np.log(prod)
def normal_params(x):
    N     = len(x)
    mu    = sum(x)/N
    var   = 1/N * sum((x-mu)**2)
    return mu,var
```

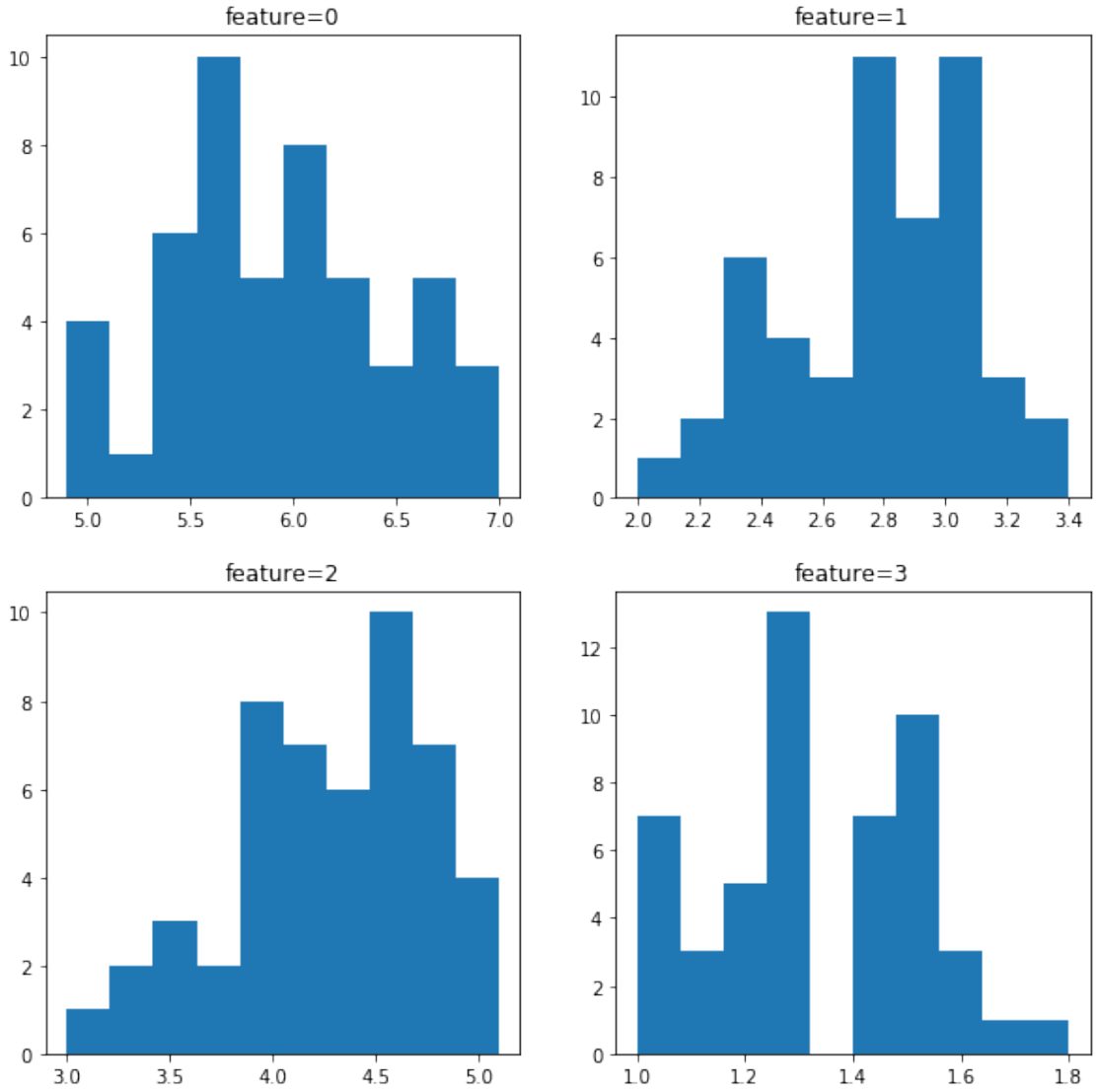
```
probs = norm.pdf(X[:,i])
print(probs)
print(np.product(probs))
```

```
[6.36982518e-06 1.59837411e-05 2.43896075e-06 1.33830226e-04
 1.01408521e-05 1.59837411e-05 6.36982518e-06 1.72256894e-03
 1.01408521e-05 1.98655471e-04 8.72682695e-04 5.89430678e-05
 1.33830226e-04 6.36982518e-06 6.11901930e-04 2.49424713e-05
 1.59837411e-05 8.92616572e-05 1.59837411e-05 1.98655471e-04
 3.96129909e-06 1.33830226e-04 2.43896075e-06 6.36982518e-06
 3.85351967e-05 2.49424713e-05 3.96129909e-06 1.48671951e-06
 1.59837411e-05 8.72682695e-04 2.91946926e-04 4.24780271e-04
 1.98655471e-04 8.97243516e-07 1.59837411e-05 1.59837411e-05
 6.36982518e-06 2.49424713e-05 8.92616572e-05 1.33830226e-04
 2.49424713e-05 1.01408521e-05 1.33830226e-04 1.72256894e-03
 5.89430678e-05 5.89430678e-05 5.89430678e-05 3.85351967e-05
 4.43184841e-03 8.92616572e-05]
4.580434206093961e-220
```

(i) Izraunajte ML-procjene za (μ, σ^2) za svaku od $n = 4$ znaajki iz skupa *Iris*. Ispiite log-izglednosti tih ML-procjena.

[558]:
```
for i in range(4):
    xs = X[:,i]
    mu,var = normal_params(xs)
    mle = normal_loglikelihood(xs, mu, var)
    print(f"feature={i},mle={mle}")

plt.figure(figsize=(10,10))
for i in range(4):
    xs = X[:,i]
    plt.subplot(2,2,i+1)
    plt.title(f"feature={i}")
    plt.hist(xs)
```

```
feature=0,mle=-74.55239091734718
feature=1,mle=-188.103670641327
feature=2,mle=-84.94239227785768
feature=3,mle=-535.1957211855389
```



Q: Moete li, na temelju dobivenih log-izglednosti, zakljuiti koja se znaajka najbolje pokorava normalnoj distribuciji?

(j) Prouite funkciju `pearsonr` za izraun Pearsonovog koeficijenta korelacije. Izraunajte koeficijente korelacije izmeu svih etiri znaajki u skupu *Iris*.

[559]:
```
from scipy.stats import pearsonr
```

[562]:
```
xs = [0,1,2,3]
for ix,c in enumerate(it.combinations(xs,2)):
    i,j = c
    xs = X[:,i]
    ys = X[:,j]
    cor,_ = pearsonr(xs,ys)
    print(f"cor(x_{i},x_{j})={cor}"
```

```
cor(x_0,x_1)=0.5259107172828247
cor(x_0,x_2)=0.7540489585920163
cor(x_0,x_3)=0.5464610715986298
cor(x_1,x_2)=0.5605220916929818
cor(x_1,x_3)=0.6639987200241114
cor(x_2,x_3)=0.7866680885228169
```

(k) Prouite funkciju `cov` te izraunajte ML-procjenu za kovarijacijsku matricu za skup *Iris*. Usporedite pristranu i nepristranu procjenu. Pokaite da se razlika (srednja apsolutna i kvadratna) smanjuje s brojem primjera (npr. isprobajte za $N/4$ i $N/2$ i N primjera).

[666]:
```
def multivariate_params(X):
    N     = len(X)
    mu    = X.sum(0)/N
    sigmas = []
    for x in X:
        x = x.reshape(1,-1)
        s = (x - mu)*(x-mu).T
        sigmas.append(s)
    sigmas = np.array(sigmas).sum(0)
    return mu,sigmas / N

def get_random_sample(X,N):
    ix = np.random.randint(len(X),size=N)
    return X[ix]

M = X.T
sigma = np.cov(M)

N  = len(X)
Ns = [N//4, N//2, N]

for n in Ns:
    X_i = get_random_sample(X, n)
    _,s = multivariate_params(X_i)
    D_abs = (abs(s-sigma)).sum()
    D_q   = ((s-sigma)**2).sum()
    print(D_abs,D_q)
    print()
```

```
0.32792409297052183 0.00978580055262791

0.25711053061224487 0.007807399586598908

0.2720197551020411 0.00758054358268057
```