SU-2019-LAB03-SVM-i-kNN

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 3: Stroj potpornih vektora i algoritam k-najbliih susjeda

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

Objavljeno: **30. rujna 2019.**
Rok za predaju: **2. prosinca 2019. u 07:00h**

0.1.2 Upute

Trea laboratorijska vjeba sastoji se od sedam zadataka. 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]:
```
import numpy as np
import pandas as pd
import mlutils
import matplotlib.pyplot as plt
%pylab inline
```

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

0.1.3 1. Klasifikator stroja potpornih vektora (SVM)

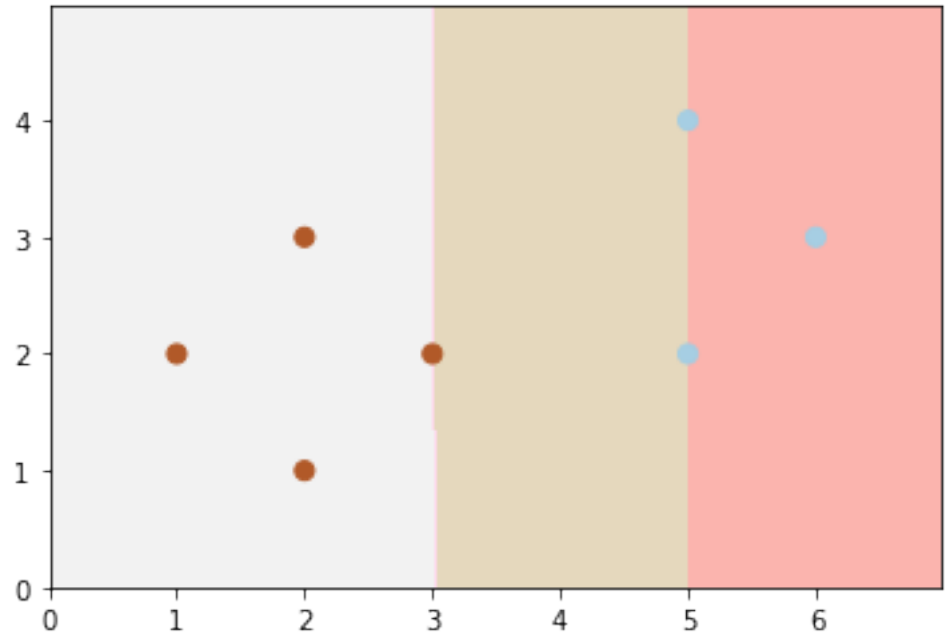
(a) Upoznajte se s razredom `svm.SVC`, koja ustvari implementira suelje prema implementaciji `libsvm`. Primijenite model `SVC` s linearnom jezgrenom funkcijom (tj. bez preslikavanja primjera u prostor znaajki) na skup podataka `seven` (dan nie) s $N = 7$ primjera. Ispiite koeficijente w_0 i $\mathbf{w}$. Ispiite dualne koeficijente i potporne vektore. Zavrno, koristei funkciju `mlutils.plot_2d_svc_problem` iscrtajte podatke, decizijsku granicu i marginu. Funkcija prima podatke, oznake i klasifikator (objekt klase `SVC`).

Izraunajte irinu dobivene margine (prisjetite se geometrije linearnih modela).

[3]:
```
from sklearn.svm import SVC

seven_X = np.array([[2,1], [2,3], [1,2], [3,2], [5,2], [5,4], [6,3]])
seven_y = np.array([1, 1, 1, 1, -1, -1, -1])
```

[4]:
```
model = SVC(kernel="linear")
model.fit(seven_X, seven_y)
mlutils.plot_2d_svc_problem(seven_X, seven_y, model)

w  = model.coef_[0]
w0 = model.intercept_
alpha = model.dual_coef_[0]
print(f"w={w}, w0={w0}, alpha={alpha}")

support_vectors = model.support_vectors_
print(f"Support vectors:\n{support_vectors}")

print(f"dual coef: {model.dual_coef_}")
```

```
w=[-9.99707031e-01 -2.92968750e-04], w0=[3.99951172], alpha=[-4.99707031e-01
-1.46484375e-04  4.99853516e-01]
Support vectors:
[[5. 2.]
 [5. 4.]
 [3. 2.]]
dual coef: [[-4.99707031e-01 -1.46484375e-04  4.99853516e-01]]
```



Q: Koliko iznosi irina margine i zato?
Q: Koji primjeri su potporni vektori i zato?

(b) Definirajte funkciju `hinge(model, x, y)` koja izraunava gubitak zglobnice modela SVM na primjeru `x`. Izraunajte gubitke modela nauenog na skupu `seven` za primjere $\mathbf{x}^{(2)} = (3,2)$ i $\mathbf{x}^{(1)} = (3.5,2)$ koji su oznaeni pozitivno ($y = 1$) te za $\mathbf{x}^{(3)} = (4,2)$ koji je oznaen negativno ($y = -1$). Takoer, izraunajte prosjeni gubitak SVM-a na skupu `seven`. Uvjerite se da je rezultat identian onome koji biste dobili primjenom ugraene funkcije `metrics.hinge_loss`.

[6]:
```
from sklearn.metrics import hinge_loss
```

[7]:
```
def hinge(model, x, y):
    h = model.predict([x]).item()
    return max(0, 1 - y*h)

X = [[3,2],
     [3.5,2],
     [4,2]]
y_true = [1,1,-1]

L = []
for x,y in zip(X,y_true):
    L.append(hinge(model, x, y))

L = np.array(L)
print(f"L={L.mean()}")
```

```
y_pred = model.predict(X)
print("sklearn={}".format(hinge_loss(y_true, y_pred)))
```

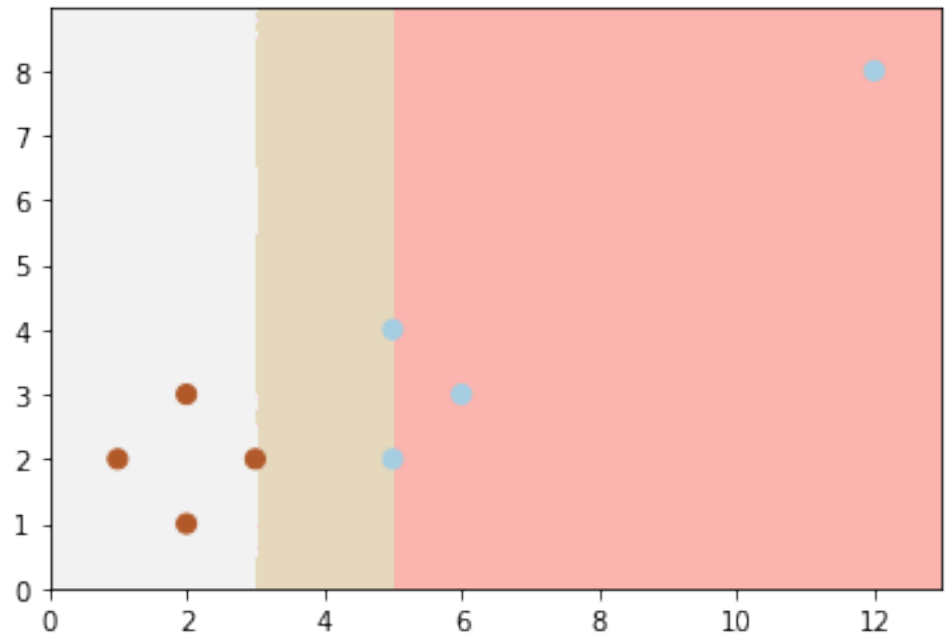
```
L=0.6666666666666666
sklearn=0.6666666666666666
```

(c) Vratit emo se na skupove podataka `outlier` ($N = 8$) i `unsep` ($N = 8$) iz prole laboratorijske vjebe (dani nie) i pogledati kako se model SVM-a nosi s njima. Nauite ugraeni model SVM-a (s linearnom jezgrom) na ovim podatcima i iscrtajte decizijsku granicu (skupa s marginom). Takoer ispiite tonost modela koritenjem funkcije `metrics.accuracy_score`.

[8]:
```
from sklearn.metrics import accuracy_score

outlier_X = np.append(seven_X, [[12,8]], axis=0)
outlier_y = np.append(seven_y, -1)

unsep_X = np.append(seven_X, [[2,2]], axis=0)
unsep_y = np.append(seven_y, -1)
```

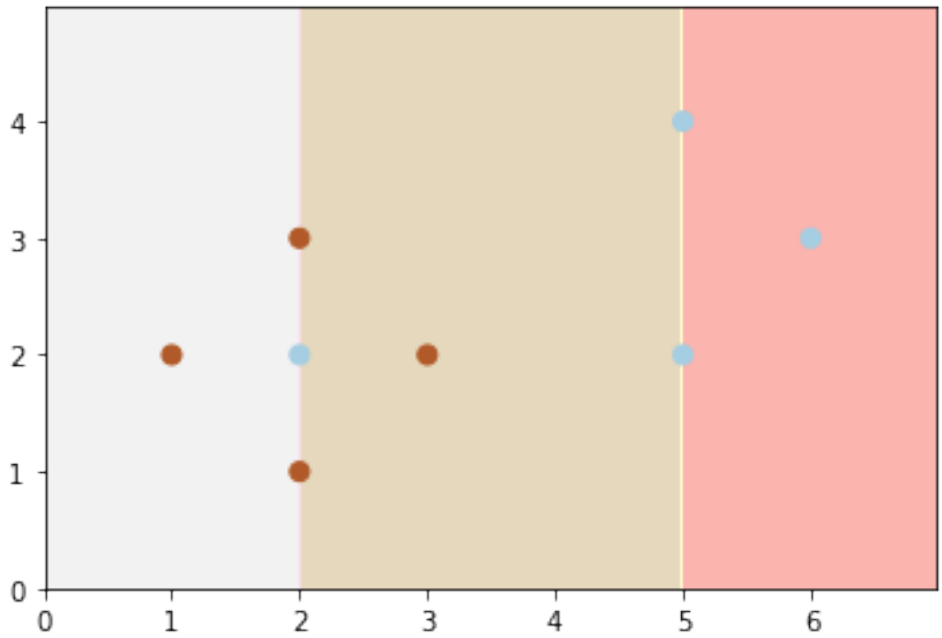
[10]:
```
model = SVC(kernel="linear").fit(outlier_X,outlier_y)
mlutils.plot_2d_svc_problem(outlier_X, outlier_y, model)
acc = accuracy_score(outlier_y, model.predict(outlier_X))
print(f"accuracy={acc}")
```

```
accuracy=1.0
```



[11]:
```
model = SVC(kernel="linear").fit(unsep_X,unsep_y)
mlutils.plot_2d_svc_problem(unsep_X, unsep_y, model)
acc = accuracy_score(unsep_y, model.predict(unsep_X))
print(acc)
```

```
0.875
```



[14]:
```
model.support_vectors_
```

[14]:
```
array([[5., 2.],
       [5., 4.],
       [2., 2.],
       [2., 3.],
       [3., 2.]])
```

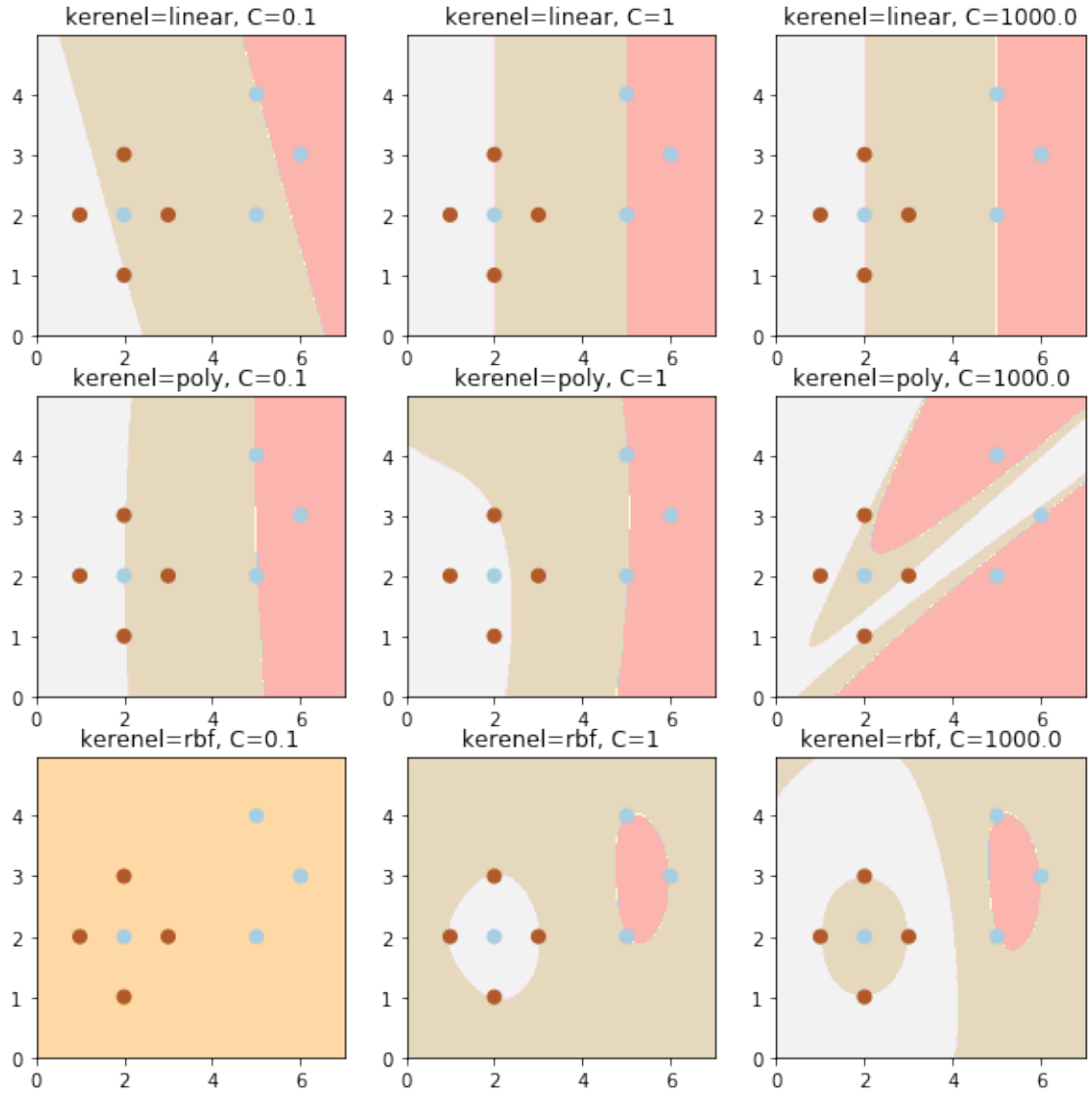
Q: Kako strea vrijednost utjee na SVM?
Q: Kako se linearan SVM nosi s linearno neodvojivim skupom podataka?

0.1.4 2. Nelinearan SVM

Ovaj zadatak pokazat e kako odabir jezgre utjee na kapacitet SVM-a. Na skupu `unsep` iz prolog zadatka trenirajte tri modela SVM-a s razliitim jezgrenim funkcijama: linearnom, polinomijalnom i radijalnom baznom (RBF) funkcijom. Varirajte hiperparametar C po vrijednostima $C \in \{10^{-2}, 1, 10^2\}$, dok za ostale hiperparametre (stupanj polinoma za polinomijalnu jezgru odnosno hiperparametar γ za jezgru RBF) koristite podrazumijevane vrijednosti. Prikaite granice izmeu klasa (i margine) na grafikonu organiziranome u polje $3x3$, gdje su stupci razliite jezgre, a retci razliite vrijednosti parametra C.

[9]:
```
import warnings
warnings.filterwarnings("ignore")

X = unsep_X.copy()
y = unsep_y.copy()
Cs      = [10e-2, 1, 10e2]
kernels = ["linear", "poly", "rbf"]

plt.figure(figsize = (10,10))
for i in range(3):
    for j in range(3):
        print(i,j)
        if kernels[i] == "rbf":
            model = SVC(C = Cs[j], kernel = kernels[i]).fit(X,y)
        else:
            model = SVC(C = Cs[j], kernel = kernels[i], gamma = "scale").
 ↪fit(X,y)
        index = j + 3*i + 1
        ax = plt.subplot(3,3,index)
        ax.set_title(f"kerenel={kernels[i]}, C={Cs[j]}")
        mlutils.plot_2d_svc_problem(X, y, model)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```



0.1.5 3. Optimizacija hiperparametara SVM-a

Pored hiperparametra C, model SVM s jezgrenom funkcijom RBF ima i dodatni hiperparametar $\gamma = \frac{1}{2\sigma^2}$ (preciznost). Taj parametar takoer odreuje sloenost modela: velika vrijednost za γ znai da e RBF biti uska, primjeri e biti preslikani u prostor u kojem su (prema skalarnome produktu) meusobno vrlo razliiti, to e rezultirati sloenijim modelima. Obrnuto, mala vrijednost za γ znai da e RBF biti iroka, primjeri e biti meusobno sliniji, to e rezultirati jednostavnijim modelima. To ujedno znai da, ako odabremo vei γ, trebamo jae regularizirati model, tj. trebamo odabrati manji C, kako bismo sprijeili prenauenost. Zbog toga je potrebno zajedniki optimirati hiperparametre C i γ, to se tipino radi iscrpnim pretraivanjem po reetci (engl. *grid search*). Ovakav pristup primjenjuje se kod svih modela koji sadre vie od jednog hiperparametra.

(a) Definirajte funkciju

```
grid_search(X_train, X_validate, y_train, y_validate, c_range=(c1,c2),
g_range=(g1,g2), error_surface=False)
```

koja optimizira parametre C i γ pretraivanjem po reetci. Funkcija treba pretraiti hiperparametre $C \in \{2^{c_1}, 2^{c_1+1}, \ldots, 2^{c_2}\}$ i $\gamma \in \{2^{g_1}, 2^{g_1+1}, \ldots, 2^{g_2}\}$. Funkcija treba vratiti optimalne hiperparametre (C^*, γ^*), tj. one za koje na skupu za provjeru model ostvaruju najmanju pogreku. Dodatno, ako je `surface=True`, funkcija treba vratiti matrice (tipa `ndarray`) pogreke modela (oekivanje gubitka 0-1) na skupu za uenje i skupu za provjeru. Svaka je matrica dimenzija $(c_2 - c_1 + 1) \times (g_2 - g_1 + 1)$ (retci odgovaraju razliitim vrijednostima za C, a stupci razliitim vrijednostima za γ).

[10]:
```
from sklearn.metrics import accuracy_score, zero_one_loss
import pdb

def accuracy(y_pred, y):
    tmp = (y_pred == y)
    N = len(y_pred)
    return 1 - tmp.sum()/N

def grid_search(X_train, X_validate, y_train, y_validate, c_range=(0,5),␣
 ↪g_range=(0,5), error_surface=False):
    Cs = [2**i for i in range(*c_range)]
    Gs = [2**i for i in range(*g_range)]

    test_errors = []
    train_errors  = np.zeros((len(Cs), len(Gs)))

    for i in range(len(Cs)):
        for j in range(len(Gs)):
            model  = SVC(C = Cs[i], gamma = Gs[j], kernel = "rbf").fit(X_train,␣
 ↪y_train)
            y_pred = model.predict(X_validate)
            e_test = accuracy(y_pred, y_validate)
            item = (e_test, Cs[i], Gs[j])
            test_errors.append(item)
            if error_surface:
                y_pred = model.predict(X_train)
                e_train = accuracy(y_pred, y_train)
                train_errors[i,j] = e_train
    best_coeffs = min(test_errors, key=lambda x : x[0])[1:]
    if error_surface:
        test_errors = list(map(lambda x : x[0], test_errors))
        test_errors = np.array(test_errors).reshape(len(Cs), len(Gs))
        return best_coeffs, train_errors, test_errors
    else:
        return best_coeffs
```

(b) Pomou funkcije `datasets.make_classification` generirajte **dva** skupa podataka od $N = 200$ primjera: jedan s $n = 2$ dimenzije i drugi s $n = 100$ dimenzija. Primjeri neka dolaze iz dviju klasa, s time da svakoj klasi odgovaraju dvije grupe (`n_clusters_per_class=2`), kako bi problem bio neto sloeniji, tj. nelinearniji. Neka sve znaajke budu informativne. Podijelite skup primjera na skup za uenje i skup za ispitivanje u omjeru 1:1.

Na oba skupa optimirajte SVM s jezgrenom funkcijom RBF, u reetci $C \in \{2^{-5}, 2^{-4}, \ldots, 2^{15}\}$ i $\gamma \in \{2^{-15}, 2^{-14}, \ldots, 2^{3}\}$. Prikaite povrinu pogreke modela na skupu za uenje i skupu za provjeru, i to na oba skupa podataka (ukupno etiri grafikona) te ispiite optimalne kombinacije hiperparametara. Za prikaz povrine pogreke modela moete koristiti funkciju `mlutils.plot_error_surface`.

[11]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

[15]:
```
n = 2
X_one, y_one = make_classification(n_samples=200,
                                   n_features=n,
                                   n_informative=n,
                                   n_redundant=0,
                                   n_classes=2,
                                   n_clusters_per_class=2)

n = 100
X_two, y_two = make_classification(n_samples=200,
                                   n_features=n,
                                   n_informative=n,
                                   n_redundant=0,
                                   n_classes=2,
                                   n_clusters_per_class=2)

Cs = (-5,15)
Gs = (-15,3)

X_one_train, X_one_test, y_one_train, y_one_test = train_test_split(X_one,␣
 ↪y_one, test_size = 0.5)
coef, e_train1, e_test1 = grid_search(X_one_train, X_one_test, y_one_train,␣
 ↪y_one_test, Cs, Gs, True)
print(f"C={coef[0]}, G={coef[1]}")

X_two_train, X_two_test, y_two_train, y_two_test = train_test_split(X_two,␣
 ↪y_two, test_size = 0.5)
coeff, e_train2, e_test2 = grid_search(X_two_train, X_two_test, y_two_train,␣
 ↪y_two_test, Cs, Gs, True)
print(f"C={coef[0]}, G={coef[1]}")

plt.figure(figsize = (10,10))

ax = plt.subplot(2,2,1)
ax.set_title("data 1, train set")
```

```
mlutils.plot_error_surface(e_train1, Cs, Gs)

ax = plt.subplot(2,2,2)
ax.set_title("data 1, test set")
mlutils.plot_error_surface(e_test1, Cs, Gs)

ax = plt.subplot(2,2,3)
ax.set_title("data 2, train set")
mlutils.plot_error_surface(e_train2, Cs, Gs)

ax = plt.subplot(2,2,4)
ax.set_title("data 2, test set")
mlutils.plot_error_surface(e_test2, Cs, Gs)
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-15-7286bc985bb6> in <module>
  1 n = 2
----> 2 X_one, y_one = make_classification(n_samples=200,
  3                                        n_features=n,
  4                                        n_informative=n,
  5                                        n_redundant=0,


NameError: name 'make_classification' is not defined
```

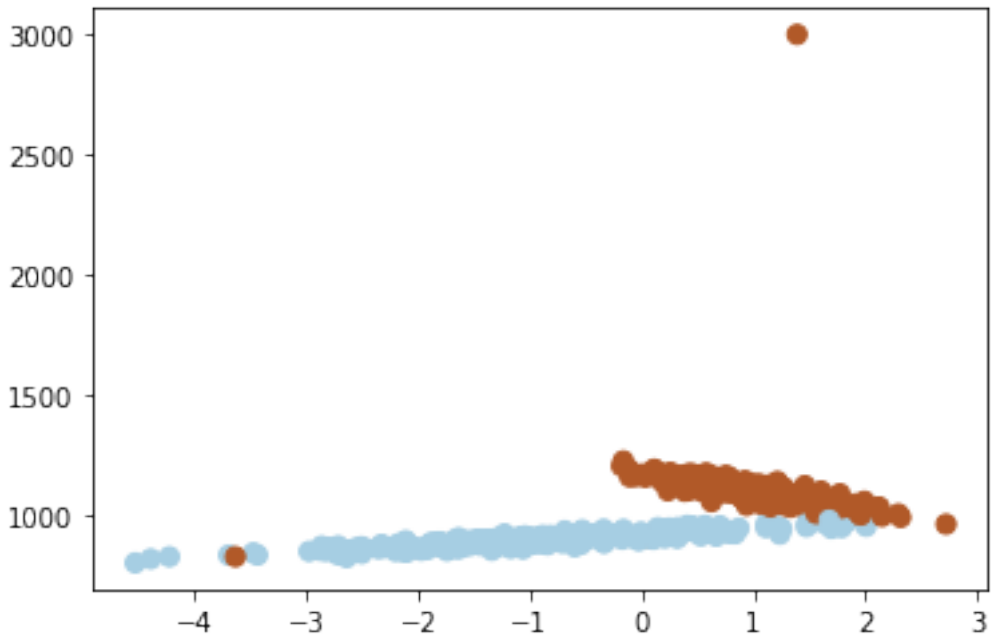
Q: Razlikuje li se povrina pogreke na skupu za uenje i skupu za ispitivanje? Zato?
Q: U prikazu povrine pogreke, koji dio povrine odgovara prenauenosti, a koji podnauenosti? Zato?
Q: Kako broj dimenzija n utjee na povrinu pogreke, odnosno na optimalne hiperparametre (C^*, γ^*)?
Q: Preporuka je da poveanje vrijednosti za γ treba biti popraeno smanjenjem vrijednosti za C. Govore li vai rezultati u prilog toj preporuci? Obrazloite.

0.1.6 4. Utjecaj standardizacije znaajki kod SVM-a

U prvoj laboratorijskoj vjebi smo pokazali kako znaajke razliitih skala mogu onemoguiti interpretaciju nauenog modela linearne regresije. Meutim, ovaj problem javlja se kod mnogih modela pa je tako skoro uvijek bitno prije treniranja skalirati znaajke, kako bi se sprijeilo da znaajke s veim numerikim rasponima dominiraju nad onima s manjim numerikim rasponima. To vrijedi i za SVM, kod kojega skaliranje nerijetko moe znatno poboljati rezultate. Svrha ovog zadataka jest eksperimentalno utvrditi utjecaj skaliranja znaajki na tonost SVM-a.

Generirat emo dvoklasni skup od $N = 500$ primjera s $n = 2$ znaajke, tako da je dimenzija x_1 veeg iznosa i veeg raspona od dimenzije x_0, te emo dodati jedan primjer koji vrijednou znaajke x_1 odskae od ostalih primjera:

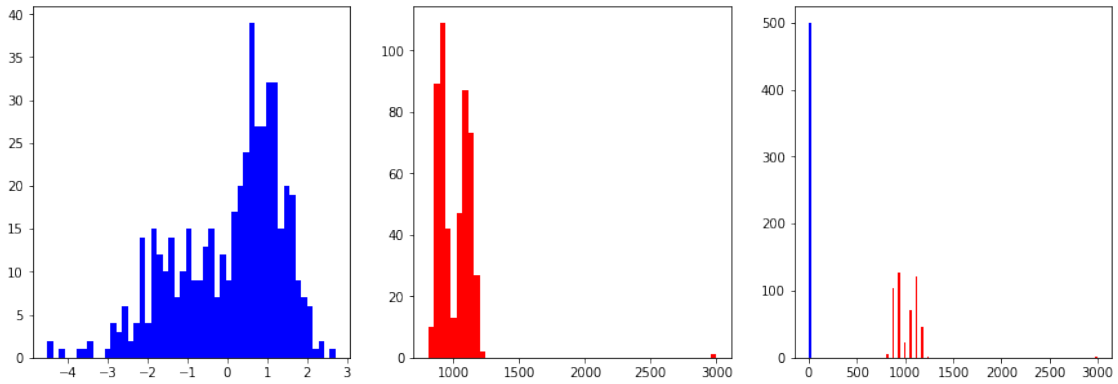
[13]:
```
from sklearn.datasets import make_classification

X, y =␣
 ↪make_classification(n_samples=500,n_features=2,n_classes=2,n_redundant=0,n_clusters_per_cla
 ↪random_state=69)
X[:,1] = X[:,1]*100+1000
X[0,1] = 3000

mlutils.plot_2d_svc_problem(X, y)
```



(a) Prouite funkciju za iscrtavanje histograma `hist`. Prikaite histograme vrijednosti znaajki x_0 i x_1 (ovdje i u sljedeim zadatcima koristite `bins=50`).

[14]:
```
def plot_hist(X):
    plt.figure(figsize=(15,5))

    plt.subplot(1,3,1)
    _ = plt.hist(X[:,0],bins = 50, color="b")
    plt.subplot(1,3,2)
    _ = plt.hist(X[:,1],bins = 50, color="r")
    plt.subplot(1,3,3)
    _ = plt.hist(X,bins = 50, color="br")
```

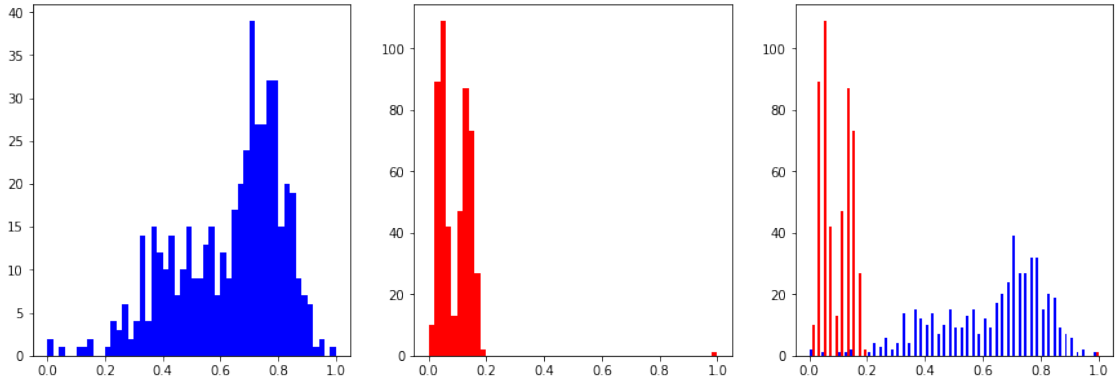
```
plot_hist(X)
```



(b) Prouite razred `preprocessing.MinMaxScaler`. Prikaite histograme vrijednosti znaajki x_0 i x_1 ako su iste skalirane min-max skaliranjem (ukupno dva histograma).

[15]:
```
from sklearn.preprocessing import MinMaxScaler
```

[16]:
```
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

plot_hist(X_scaled)
```

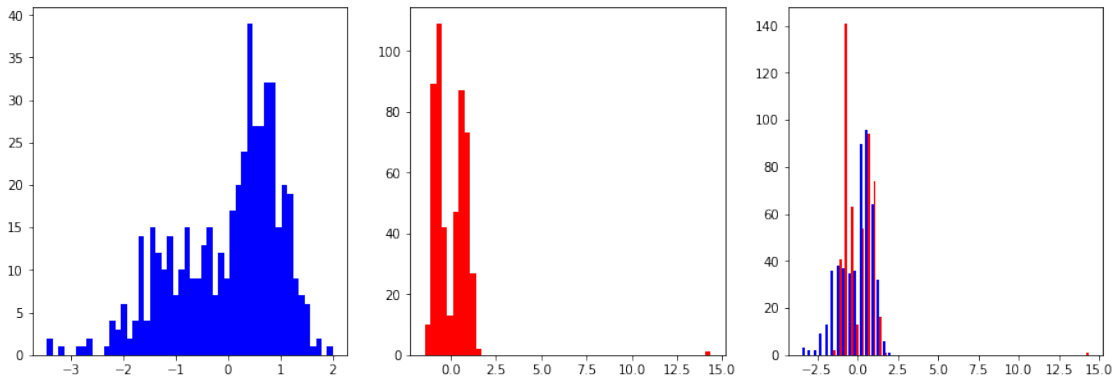


Q: Kako radi ovo skaliranje? **Q:** Dobiveni histogrami su vrlo slini. U emu je razlika?

(c) Prouite razred `preprocessing.StandardScaler`. Prikaite histograme vrijednosti znaajki x_0 i x_1 ako su iste skalirane standardnim skaliranjem (ukupno dva histograma).

[17]:
```
from sklearn.preprocessing import StandardScaler
```

[18]:
```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

plot_hist(X_scaled)
```



Q: Kako radi ovo skaliranje? **Q:** Dobiveni histogrami su vrlo slini. U emu je razlika?

(d) Podijelite skup primjera na skup za uenje i skup za ispitivanje u omjeru 1:1. Trenirajte SVM s jezgrenom funkcijom RBF na skupu za uenje i ispitajte tonost modela na skupu za ispitivanje, koristei tri varijante gornjeg skupa: neskalirane znaajke, standardizirane znaajke i min-max skaliranje. Koristite podrazumijevane vrijednosti za C i γ. Izmjerite tonost svakog od triju modela na skupu za uenje i skupu za ispitivanje. Ponovite postupak vie puta (npr. 30) te uprosjeite rezultate (u svakom ponavljanju generirajte podatke kao to je dano na poetku ovog zadatka).

NB: Na skupu za uenje treba najprije izraunati parametre skaliranja te zatim primijeniti skaliranje (funkcija `fit_transform`), dok na skupu za ispitivanje treba samo primijeniti skaliranje s parametrima koji su dobiveni na skupu za uenje (funkcija `transform`).

[19]:
```
def standard_scaler(X, ret_scaler=False):
    scaler = StandardScaler()
    scaler.fit(X)
    if ret_scaler:
        return scaler, scaler.transform(X)
    else:
        return scaler.transform(X)
def min_max_scaler(X, ret_scaler=False):
    scaler = MinMaxScaler()
    scaler.fit(X)
    if ret_scaler:
        return scaler, scaler.transform(X)
    else:
        return scaler.transform(X)

errors = {"test_normal"  : [],
```

```
          "train_normal" : [],
          "test_mm"      : [],
          "train_mm"     : [],
          "train"        : [],
          "test"         : []
         }

for i in range(30):
    X, y =␣
 ↪make_classification(n_samples=500,n_features=2,n_classes=2,n_redundant=0,n_clusters_per_cla
 ↪random_state=69)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)

    model   = SVC(kernel = "rbf").fit(X_train, y_train)
    train_acc = accuracy_score(model.predict(X_train), y_train)
    test_acc  = accuracy_score(model.predict(X_test), y_test)

    normal_scaler, X_normal = standard_scaler(X_train, True)
    model  = SVC(kernel = "rbf").fit(X_normal, y_train)
    train_normal_acc = accuracy_score(model.predict(X_normal), y_train)
    X_normal_test = normal_scaler.transform(X_test)
    test_normal_acc  = accuracy_score(model.predict(X_normal_test), y_test)

    mm_scaler, X_mm = min_max_scaler(X_train,True)
    model  = SVC(kernel = "rbf").fit(X_mm, y_train)
    train_mm_acc = accuracy_score(model.predict(X_mm), y_train)
    X_mm_test = mm_scaler.transform(X_test)
    test_mm_acc  = accuracy_score(model.predict(X_mm_test), y_test)

    errors["test_normal"].append(test_normal_acc)
    errors["train_normal"].append(train_normal_acc)
    errors["test_mm"].append(test_mm_acc)
    errors["train_mm"].append(train_mm_acc)
    errors["test"].append(test_acc)
    errors["train"].append(train_acc)


for name in errors:
    xs   = errors[name]
    mean = sum(xs)/len(xs)
    errors[name] = mean

for name in errors:
    value   = errors[name]
    print(f"{name} : {value}")
```

```
test_normal : 0.9919999999999999
```

```
train_normal : 0.9954666666666666
test_mm : 0.9874666666666666
train_mm : 0.9881333333333333
train : 0.9953333333333332
test : 0.9933333333333333
```

Q: Jesu li rezultati oekivani? Obrazloite. **Q:** Bi li bilo dobro kada bismo funkciju `fit_transform` primijenili na cijelom skupu podataka? Zato? Bi li bilo dobro kada bismo tu funkciju primijenili zasebno na skupu za uenje i zasebno na skupu za ispitivanje? Zato?

0.1.7 5. Algoritam k-najbliih susjeda

U ovom zadatku promatrat emo jednostavan klasifikacijski model imena **algoritam k-najbliih susjeda**. Najprije ete ga samostalno isprogramirati kako biste se detaljno upoznali s radom ovog modela, a zatim ete prijei na analizu njegovih hiperparametara (koristei ugraeni razred, radi efikasnosti).

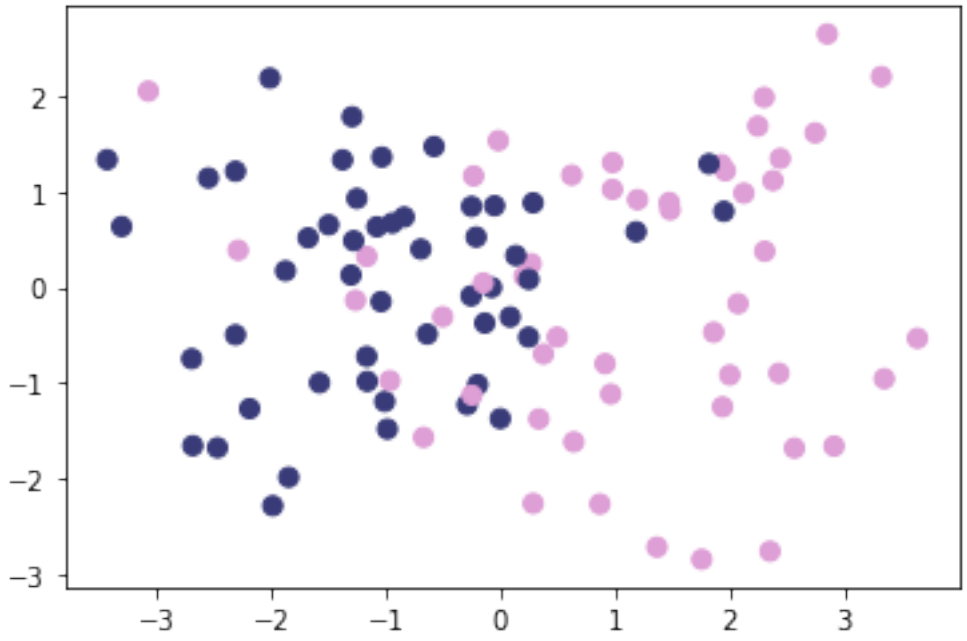
(a) Implementirajte klasu `KNN`, koja implementira algoritam k najbliih susjeda. Neobavezan parametar konstruktora jest broj susjeda `n_neighbours` (k), ija je podrazumijevana vrijednost 3. Definirajte metode `fit(X, y)` i `predict(X)`, koje slue za uenje modela odnosno predikciju. Kao mjeru udaljenosti koristite euklidsku udaljenost (`numpy.linalg.norm`; pripazite na parametar `axis`). Nije potrebno implementirati nikakvu teinsku funkciju.

[20]:
```python
from numpy.linalg import norm
from bisect import insort
from collections import Counter


class KNN:
    def __init__(self, n_neighbors=3):
        # Va kôd ovdje
        self.n_neighbors = n_neighbors

    def get_k_neighbours(self, xs):
        ix = []
        for i in range(self.n_neighbors):
            ind = np.argmin(xs)
            ix.append(ind)
            xs[ind] = np.inf
        return ix

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        y_pred = []
        for x in X_test:
            norms = norm(x - self.X_train, axis = 1)
```

```
            ix = self.get_k_neighbours(norms)
            classes = self.y_train[ix]
            c = Counter(classes)
            prediction = max(c, key = lambda x : c[x])
            y_pred.append(prediction)
        return np.array(y_pred)
```

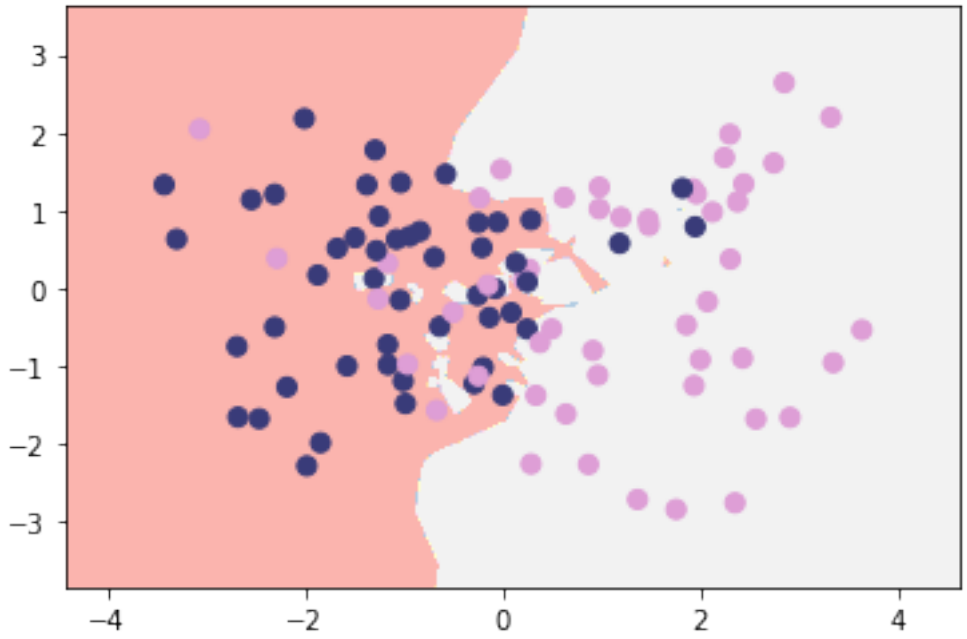
(b) Kako biste se uvjerili da je Vaa implementacija ispravna, usporedite ju s onom u razredu `neighbors.KNeighborsClassifier`. Budui da spomenuti razred koristi razne optimizacijske trikove pri pronalasku najboljih susjeda, obavezno postavite parametar `algorithm=brute`, jer bi se u protivnom moglo dogoditi da Vam se predikcije razlikuju. Usporedite modele na danom (umjetnom) skupu podataka (prisjetite se kako se usporeuju polja; `numpy.all`).

[21]:
```
from sklearn.datasets import make_classification
X_art, y_art = make_classification(n_samples=100, n_features=2, n_classes=2,
                                   n_redundant=0, n_clusters_per_class=2,
                                   random_state=69)
mlutils.plot_2d_clf_problem(X_art, y_art)
```

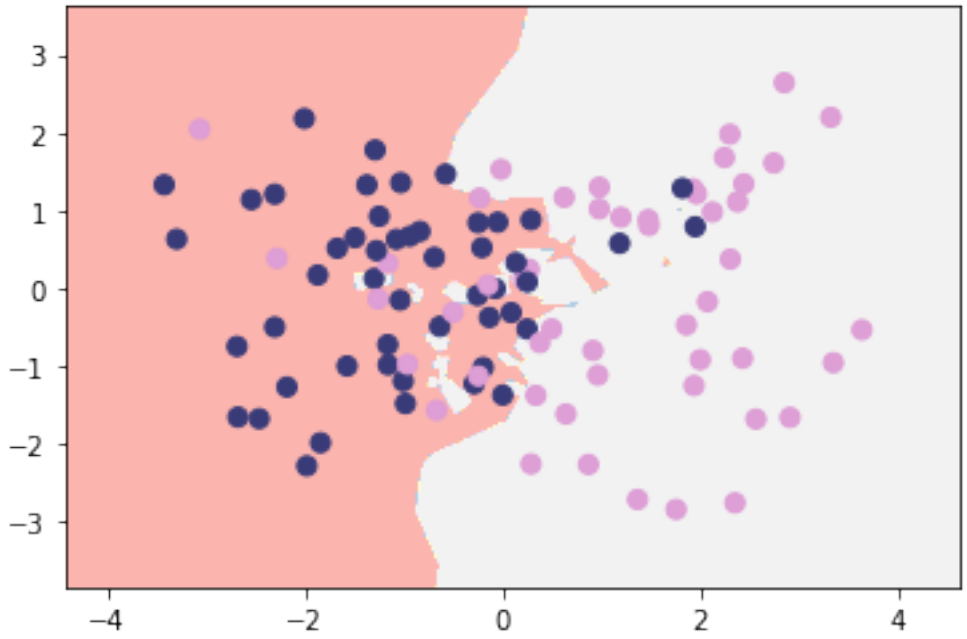


[22]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[23]:
```
knn = KNeighborsClassifier(n_neighbors=3, algorithm="brute")
knn.fit(X_art, y_art)

mlutils.plot_2d_clf_problem(X_art, y_art, h = knn.predict)
```



[24]:
```
knn = KNN(3)
knn.fit(X_art, y_art)

mlutils.plot_2d_clf_problem(X_art, y_art, h = knn.predict)
```



0.1.8 6. Analiza algoritma k-najbliih susjeda

Algoritam k-nn ima hiperparametar k (broj susjeda). Taj hiperparametar izravno utjee na sloenost algoritma, pa je stoga izrazito vano dobro odabrati njegovu vrijednost. Kao i kod mnogih drugih algoritama, tako i kod algoritma k-nn optimalna vrijednost hiperametra k ovisi o konkretnom problemu, ukljuivo broju primjera N, broju znaajki (dimenzija) n te broju klasa K.

Kako bismo dobili pouzdanije rezultate, potrebno je neke od eksperimenata ponoviti na razliitim skupovima podataka i zatim uprosjeiti dobivene vrijednosti pogreaka. Koristite funkciju: `mlutils.knn_eval` koja trenira i ispituje model k-najbliih susjeda na ukupno `n_instances` primjera, i to tako da za svaku vrijednost hiperparametra iz zadanog intervala `k_range` ponovi `n_trials` mjerenja, generirajui za svako od njih nov skup podataka i dijelei ga na skup za uenje i skup za ispitivanje. Udio skupa za ispitivanje definiran je parametrom `test_size`. Povratna vrijednost funkcije jest etvorka `(ks, best_k, train_errors, test_errors)`. Vrijednost `best_k` je optimalna vrijednost hiperparametra k (vrijednost za koju je pogreka na skupu za ispitivanje najmanja). Vrijednosti `train_errors` i `test_errors` liste su pogreaka na skupu za uenja odnosno skupu za testiranje za sve razmatrane vrijednosti hiperparametra k, dok `ks` upravo pohranjuje sve razmatrane vrijednosti hiperparametra k.

(a) Na podatcima iz zadatka 5, pomou funkcije `mlutils.plot_2d_clf_problem` iscrtajte prostor primjera i podruja koja odgovaraju prvoj odnosno drugoj klasi. Ponovite ovo za $k \in [1, 5, 20, 100]$.

NB: Implementacija algoritma `KNeighborsClassifier` iz paketa `scikit-learn` vjerojatno e raditi bre od Vae implementacije, pa u preostalim eksperimentima koristite nju.

[25]:
```
ks = [1, 5, 20, 99]
plt.figure(figsize=(20,20))
for i,k in enumerate(ks):
    print(f"iter={i}")
    knn = KNeighborsClassifier(n_neighbors=k, algorithm="brute")
    knn.fit(X_art, y_art)
    plt.subplot(2,2,i+1)
    mlutils.plot_2d_clf_problem(X_art, y_art, h = knn.predict)
```

```
iter=0
iter=1
iter=2
iter=3
```

Q: Kako k utjee na izgled granice izmeu klasa?
Q: Kako se algoritam ponaa u ekstremnim situacijama: $k = 1$ i $k = 100$?

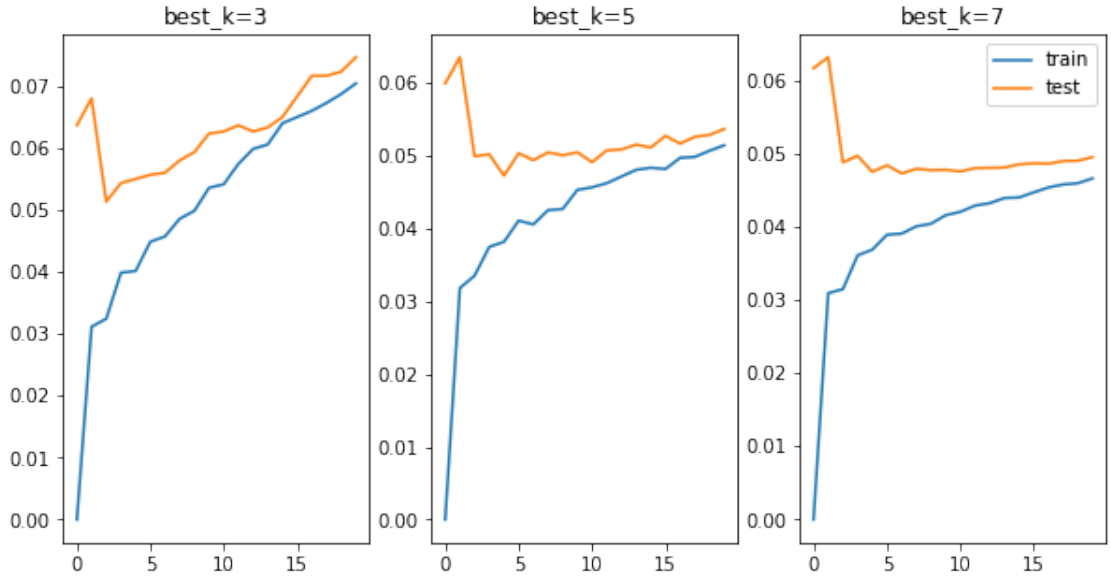
(b) Pomou funkcije `mlutils.knn_eval`, iscrtajte pogreke uenja i ispitivanja kao funkcije hiperparametra $k \in \{1, \ldots, 20\}$, za $N = \{100, 250, 750\}$ primjera. Nainite 3 zasebna grafikona. Za svaki ispiite optimalnu vrijednost hiperparametra k (najlake kao naslov grafikona; vidi `plt.title`).

[26]:
```
N = [100, 250, 750]
plt.figure(figsize=(10,5))
for i,n_instances in enumerate(N):
    print(f"iter={i}")
    ks, best_k, train_errors, test_errors = mlutils.
 ↪knn_eval(n_instances=n_instances)
    plt.subplot(1,3,i+1)
    plt.title(f"best_k={best_k}")
```

```
    plt.plot(train_errors,label="train")
    plt.plot(test_errors, label="test")

plt.legend()
```

```
iter=0
iter=1
iter=2
```

[26]: `<matplotlib.legend.Legend at 0x7fe4780b64e0>`



Q: Kako se mijenja optimalna vrijednost hiperparametra k s obzirom na broj primjera N? Zato?
Q: Kojem podruju odgovara prenauenost, a kojem podnauenost modela? Zato?
Q: Je li uvijek mogue dosei pogreku od 0 na skupu za uenje?

(c) Kako bismo provjerili u kojoj je mjeri algoritam k-najbliih susjeda osjetljiv na prisustvo nebitnih znaajki, moemo iskoristiti funkciju `datasets.make_classification` kako bismo generirali skup primjera kojemu su neke od znaajki nebitne. Naime, parametar `n_informative` odreuje broj bitnih znaajki, dok parametar `n_features` odreuje ukupan broj znaajki. Ako je `n_features > n_informative`, onda e neke od znaajki biti nebitne. Umjesto da izravno upotrijebimo funkciju `make_classification`, upotrijebit emo funkciju `mlutils.knn_eval`, koja samo preuzime ove parametre, ali nam omoguuje pouzdanije procjene.

Koristite funkciju `mlutils.knn_eval` na dva naina. U oba koristite $N = 1000$ primjera, $n = 10$ znaajki i $K = 5$ klasa, ali za prvi neka su svih 10 znaajki bitne, a za drugi neka je bitno samo 5 od 10 znaajki. Ispiite pogreke uenja i ispitivanja za oba modela za optimalnu vrijednost k (vrijednost za koju je ispitna pogreka najmanja).

[ ]:
```
ks1, best_k1, train_errors1, test_errors1 = mlutils.
 ↪knn_eval(n_instances=1000,n_features=10,n_classes=5,n_informative=10)
ks2, best_k2, train_errors2, test_errors2 = mlutils.
 ↪knn_eval(n_instances=1000,n_features=10,n_classes=5,n_informative=5)

print(f"train={train_errors1[best_k1-1]}, test={test_errors1[best_k1-1]}")
print(f"train={train_errors2[best_k2-1]}, test={test_errors2[best_k2-1]}")

plt.subplot(1,2,1)
plt.plot(train_errors1,label="train")
plt.plot(test_errors1, label="test")

plt.subplot(1,2,2)
plt.plot(train_errors2,label="train")
plt.plot(test_errors2, label="test")

plt.legend()
```

Q: Je li algoritam k-najbliih susjeda osjetljiv na nebitne znaajke? Zato?
Q: Je li ovaj problem izraen i kod ostalih modela koje smo dosad radili (npr. logistika regresija)?
Q: Kako bi se model k-najbliih susjeda ponaao na skupu podataka sa znaajkama razliitih skala? Detaljno pojasnite.

0.1.9 7. "Prokletstvo dimenzionalnosti"

"Prokletstvo dimenzionalnosti" zbirni je naziv za niz fenomena povezanih s visokodimenzijskim prostorima. Ti fenomeni, koji se uglavnom protive naoj intuiciji, u veini sluajeva dovode do toga da se s porastom broja dimenzija (znaajki) smanjenje tonost modela.

Openito, poveanje dimenzija dovodi do toga da sve toke u ulaznome prostoru postaju (u smislu euklidske udaljenosti) sve udaljenije jedne od drugih te se, posljedino, gube razlike u udaljenostima izmeu toaka. Eksperimentalno emo provjeriti da je to doista sluaj. Prouite funkciju `metrics.pairwise_distances`. Generirajte 100 sluajnih vektora u razliitim dimenzijama $n \in [1, 2, \ldots, 50]$ dimenzija te izraunajte *prosjenu* euklidsku udaljenost izmeu svih parova tih vektora. Za generiranje sluajnih vektora koristite funkciju `numpy.random.random`. Na istom grafu skicirajte i krivulju za prosjene kosinusne udaljenosti (parametar `metric`).

[ ]:
```
from sklearn.metrics.pairwise import pairwise_distances
```

[ ]:
```
dist1 = []
dist2 = []
dims = list(range(1,50))
for d in dims:
    X = np.random.random((100,d))
    D = pairwise_distances(X)
    dist1.append(D.mean())
    D = pairwise_distances(X,metric="cosine")
    dist2.append(D.mean())

plt.plot(dims, dist1,label = "euclidian")
```

```
plt.plot(dims, dist2, label = "cosine")
plt.legend()
```

Q: Pokuajte objasniti razlike u rezultatima. Koju biste od ovih dviju mjera koristili za klasifikaciju visokodimenzijskih podataka?
Q: Zato je ovaj problem osobito izraen kod algoritma k-najbliih susjeda?