Support Vector Machine and k-Nearest Neighbours

February 5, 2021

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

0.1 Strojno učenje 2020/2021

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

0.1.1 Laboratorijska vježba 3: Stroj potpornih vektora i algoritam k-najbližih susjeda

Verzija: 0.5
Zadnji put ažurirano: 30. studenog 2020.

Rok za predaju: **7. prosinca 2020. u 06:00h**

0.1.2 Upute

Treća laboratorijska vježba sastoji se od sedam zadataka. U nastavku slijedite upute navedene u ćelijama s tekstom. Rješavanje vježbe svodi se na **dopunjavanje ove bilježnice**: umetanja ćelije ili više njih **ispod** teksta zadatka, pisanja odgovarajućeg kôda te evaluiranja ćelija.

Osigurajte da u potpunosti **razumijete** kôd koji ste napisali. Kod predaje vježbe, morate biti u stanju na zahtjev asistenta (ili demonstratora) preinačiti i ponovno evaluirati Vaš kôd. Nadalje, morate razumjeti teorijske osnove onoga što radite, u okvirima onoga što smo obradili na predavanju. Ispod nekih zadataka možete naći i pitanja koja služe kao smjernice za bolje razumijevanje gradiva (**nemojte pisati** odgovore na pitanja u bilježnicu). Stoga se nemojte ograničiti samo na to da riješite zadatak, nego slobodno eksperimentirajte. To upravo i jest svrha ovih vježbi.

Vježbe trebate raditi **samostalno** ili u **tandemu**. Možete se konzultirati s drugima o načelnom načinu rješavanja, ali u konačnici morate sami odraditi vježbu. U protivnome vježba nema smisla.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
%pylab inline
```

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

[2]:
```
def plot_2d_clf_problem(X, y, h=None):
    '''
    Plots a two-dimensional labeled dataset (X,y) and, if function h(x) is␣
 ↪given,
    the decision surfaces.
    '''
    assert X.shape[1] == 2, "Dataset is not two-dimensional"
    if h!=None :
        # Create a mesh to plot in
        r = 0.03  # mesh resolution
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, r),
                             np.arange(y_min, y_max, r))
        XX=np.c_[xx.ravel(), yy.ravel()]
        try:
            Z_test = h(XX)
            if Z_test.shape == ():
                # h returns a scalar when applied to a matrix; map explicitly
                Z = np.array(list(map(h,XX)))
            else :
                Z = Z_test
        except ValueError:
            # can't apply to a matrix; map explicitly
            Z = np.array(list(map(h,XX)))
        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.Pastel1)

    # Plot the dataset
    plt.scatter(X[:,0],X[:,1], c=y, cmap=plt.cm.tab20b, marker='o', s=50);

def plot_2d_svc_problem(X, y, svc=None):
    '''
    Plots a two-dimensional labeled dataset (X,y) and, if SVC object is given,
    the decision surfaces (with margin as well).
    '''
    assert X.shape[1] == 2, "Dataset is not two-dimensional"
    if svc!=None :
        # Create a mesh to plot in
        r = 0.03  # mesh resolution
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, r),
                             np.arange(y_min, y_max, r))
        XX=np.c_[xx.ravel(), yy.ravel()]
        Z = np.array([svc_predict(svc, x) for x in XX])
```

```
        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.Pastel1)

    # Plot the dataset
    plt.scatter(X[:,0],X[:,1], c=y, cmap=plt.cm.Paired, marker='o', s=50)

def svc_predict(svc, x) :
    h = svc.decision_function([x])
    if np.isclose(h, 0, atol=0.03):
        return 5
    elif (h >= -1 and h < -0.03) or (h > 0.03 and h <= 1):
        return 0.5
    else:
        return max(-1, min(1, h))

def plot_error_surface(err, c_range=(0,5), g_range=(0,5)):
    c1, c2 = c_range[0], c_range[1]
    g1, g2 = g_range[0], g_range[1]
    plt.xticks(range(0,g2-g1+1,5),range(g1,g2+1,5)); plt.xlabel("gamma")
    plt.yticks(range(0,c2-c1+1,5),range(c1,c2+1,5)); plt.ylabel("C")
    p = plt.contour(err);
    plt.imshow(1-err, interpolation='bilinear', origin='lower',cmap=plt.cm.gray)
    plt.clabel(p, inline=1, fontsize=10)

def knn_eval(n_instances=100, n_features=2, n_classes=2, n_informative=2,
             test_size=0.3, k_range=(1, 20), n_trials=40):

    train_errors = []
    test_errors = []
    ks = list(range(k_range[0], k_range[1] + 1))

    for i in range(0, n_trials):
        X, y = make_classification(n_instances, n_features, n_classes=n_classes,
                                   n_informative=n_informative, n_redundant=0,␣
↪n_clusters_per_class=1)
        X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=test_size)
        train = []
        test = []
        for k in ks:
            knn = KNeighborsClassifier(n_neighbors=k)
            knn.fit(X_train, y_train)
            train.append(1 - knn.score(X_train, y_train))
            test.append(1 - knn.score(X_test, y_test))
        train_errors.append(train)
        test_errors.append(test)
```

```
    train_errors = np.mean(np.array(train_errors), axis=0)
    test_errors = np.mean(np.array(test_errors), axis=0)
    best_k = ks[np.argmin(test_errors)]

    return ks, best_k, train_errors, test_errors
```

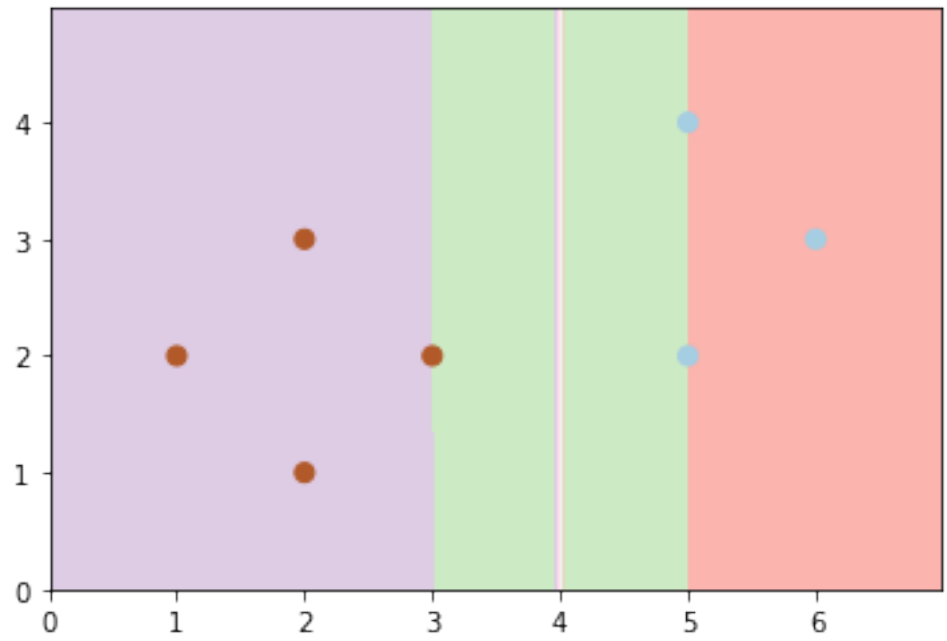
0.1.3 1. Klasifikator stroja potpornih vektora (SVM)

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

[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]:
```
machine = SVC(kernel='linear')
machine.fit(seven_X, seven_y)
prediction = machine.predict(seven_X)

intercept = machine.intercept_
weights = machine.coef_
dual_coefs = machine.dual_coef_
support_vectors = machine.support_vectors_

print("w0 = " + str(intercept))
print("w = " + str(weights))
print("dual coefficients = " + str(dual_coefs))
print("support vectors = " + str(support_vectors))

plot_2d_svc_problem(seven_X, prediction, machine)
```

```
w0 = [3.99951172]
w = [[-9.99707031e-01 -2.92968750e-04]]
dual coefficients = [[-4.99707031e-01 -1.46484375e-04  4.99853516e-01]]
support vectors = [[5. 2.]
 [5. 4.]
 [3. 2.]]
```



Q: Koji primjeri su potporni vektori i zašto?

(b) Definirajte funkciju `hinge(model, x, y)` koja izračunava gubitak zglobnice modela SVM na primjeru `x`. Izračunajte gubitke modela naučenog na skupu `seven` za primjere $\mathbf{x}^{(2)} = (3, 2)$ i $\mathbf{x}^{(1)} = (3.5, 2)$ koji su označeni pozitivno ($y = 1$) te za $\mathbf{x}^{(3)} = (4, 2)$ koji je označen negativno ($y = -1$). Također, izračunajte prosječni gubitak SVM-a na skupu `seven`. Uvjerite se da je rezultat identičan onome koji biste dobili primjenom ugrađene funkcije `metrics.hinge_loss`.

[5]:
```
from sklearn.metrics import hinge_loss

# Vaš kôd ovdje...
def hinge(model, x, y):
    prediction = model.predict([x])
    to_return = max(0, 1 - y * prediction)
    return to_return

x2 = np.array([3, 2])
y2 = np.array([1])

x1 = np.array([3.5, 2])
y1 = np.array([1])

x3 = np.array([4, 2])
y3 = np.array([-1])

new_samples = np.array([x2, x1, x3])
```

```
new_labels = np.array([y2, y1, y3])

hinge1 = hinge(machine, x1, y1)
hinge2 = hinge(machine, x2, y2)
hinge3 = hinge(machine, x3, y3)

avg = (hinge1 + hinge2 + hinge3) / 3.0

print(avg)
print(hinge_loss(new_labels, machine.predict(new_samples)))
```

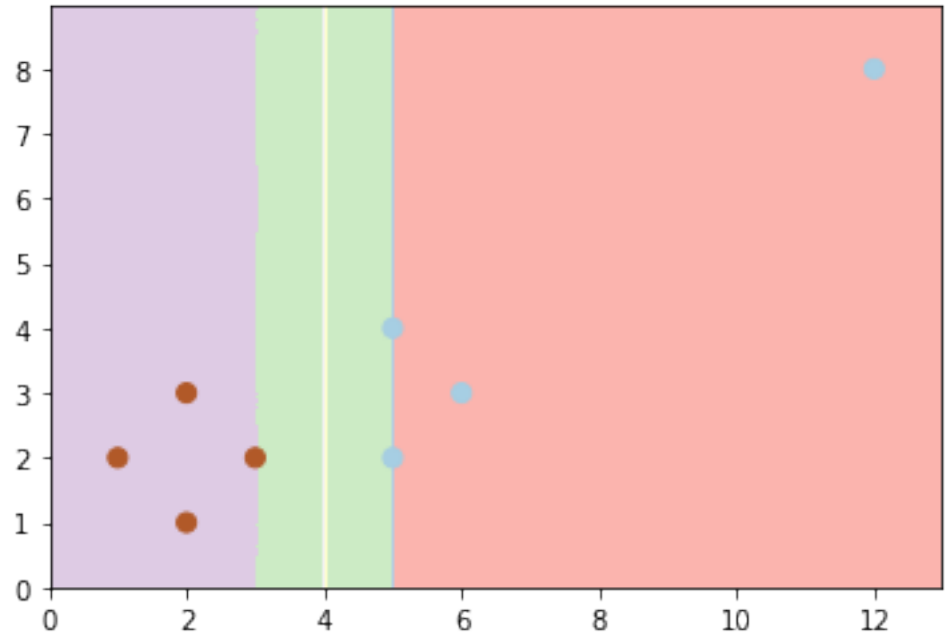
```
[0.66666667]
0.6666666666666666
```

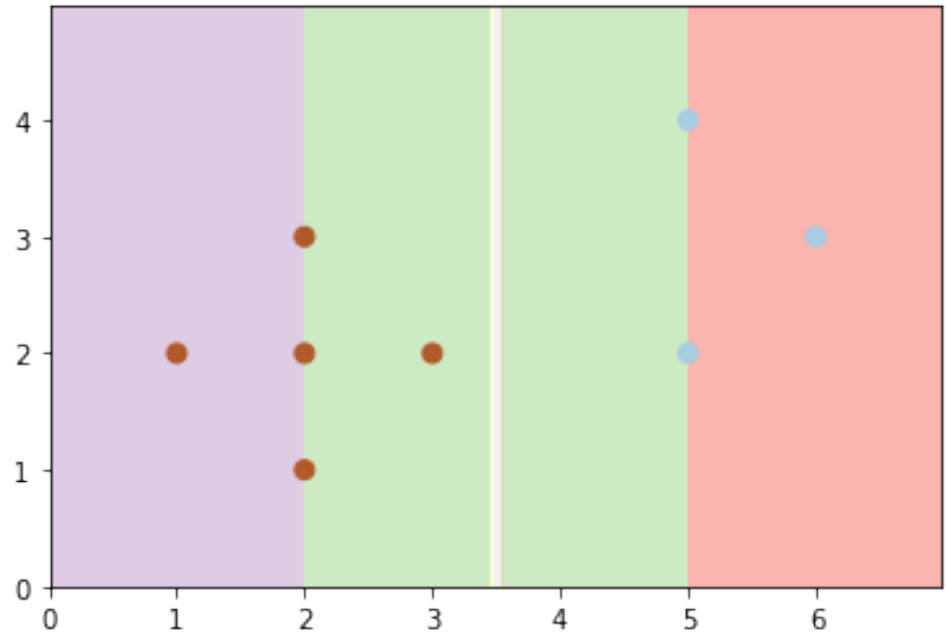
(c) Vratit ćemo se na skupove podataka `outlier` ($N = 8$) i `unsep` ($N = 8$) iz prošle laboratorijske vježbe (dani niže) i pogledati kako se model SVM-a nosi s njima. Naučite ugrađeni model SVM-a (s linearnom jezgrom) na ovim podatcima i iscrtajte decizijsku granicu (skupa s marginom). Također ispišite točnost modela korištenjem funkcije `metrics.accuracy_score`.

[6]:
```
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)
```

[7]:
```
# Vaš kôd ovdje...
linear_machine = SVC(kernel='linear')

outlier_pred = linear_machine.fit(outlier_X, outlier_y).predict(outlier_X)
plot_2d_svc_problem(outlier_X, outlier_pred, linear_machine)
print("Outlier accuracy: " + str(accuracy_score(outlier_y, outlier_pred)))

plt.figure()

unsep_pred = linear_machine.fit(unsep_X, unsep_y).predict(unsep_X)
plot_2d_svc_problem(unsep_X, unsep_pred, linear_machine)
print("Unseparable accuracy: " + str(accuracy_score(unsep_y, unsep_pred)))
```

```
Outlier accuracy: 1.0
Unseparable accuracy: 0.875
```



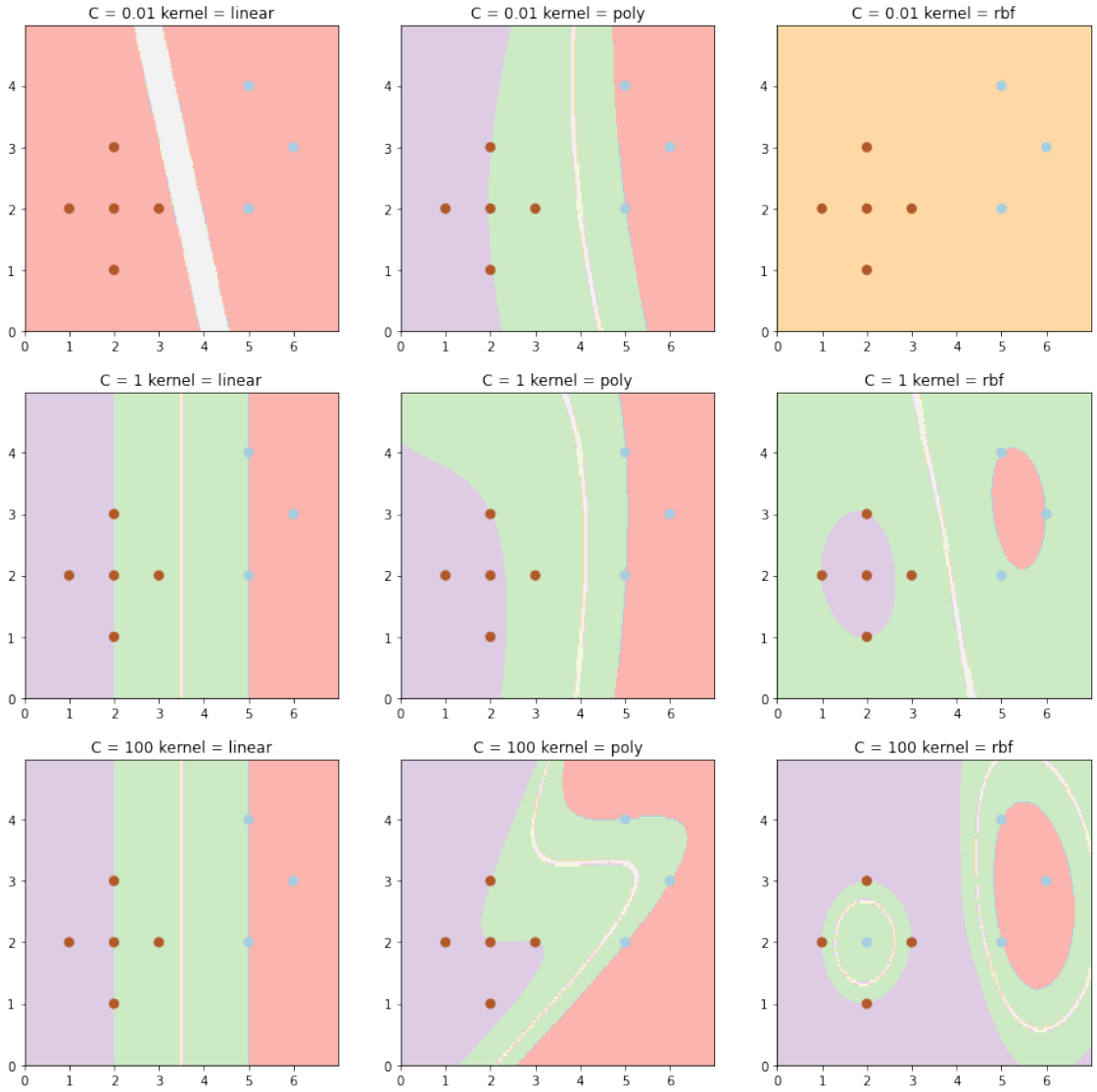


Q: Kako stršeća vrijednost utječe na SVM?
Q: Kako se linearan SVM nosi s linearno neodvojivim skupom podataka?

0.1.4 2. Nelinearan SVM

Ovaj zadatak pokazat će kako odabir jezgre utječe na kapacitet SVM-a. Na skupu `unsep` iz prošlog zadatka trenirajte tri modela SVM-a s različitim 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. Prikažite granice između klasa (i margine) na grafikonu organiziranome u polje $3x3$, gdje su stupci različite jezgre, a retci različite vrijednosti parametra C.

[8]:
```
# Vaš kôd ovdje...

iterations = [0, 1, 2]

C = [10**(-2), 1, 10**2]

kernels = ['linear', 'poly', 'rbf']

figure(figsize(15, 15))
plt.tight_layout()

for row, c in zip(iterations, C):
    for col, kernel in zip(iterations, kernels):
        machine = SVC(C=c, kernel=kernel)
        machine.fit(unsep_X, unsep_y)
        prediction = machine.predict(unsep_X)

        subplot(3, 3, row * 3 + col + 1)
        plot_2d_svc_problem(unsep_X, prediction, machine)
        title('C = ' + str(c) + ' ' + 'kernel = ' + kernel)
```



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 također određuje složenost modela: velika vrijednost za γ znači da će RBF biti uska, primjeri će biti preslikani u prostor u kojem su (prema skalarnome produktu) međusobno vrlo različiti, što će rezultirati složenijim modelima. Obrnuto, mala vrijednost za γ znači da će RBF biti široka, primjeri će biti međusobno sličniji, što će rezultirati jednostavnijim modelima. To ujedno znači da, ako odabremo veći γ, trebamo jače regularizirati model, tj. trebamo odabrati manji C, kako bismo spriječili prenaučenost. Zbog toga je potrebno zajednički optimirati hiperparametre C i γ, što se tipično radi iscrpnim pretraživanjem po rešetci (engl. *grid search*). Ovakav pristup primjenjuje se kod svih modela koji sadrže više 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 pretraživanjem po rešetci. Funkcija treba pretražiti 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 pogrešku. Dodatno, ako je `surface=True`, funkcija treba vratiti matrice (tipa `ndarray`) pogreške modela (očekivanje gubitka 0-1) na skupu za učenje i skupu za provjeru. Svaka je matrica dimenzija $(c_2 - c_1 + 1) \times (g_2 - g_1 + 1)$ (retci odgovaraju različitim vrijednostima za C, a stupci različitim vrijednostima za γ).

[9]:
```
from sklearn.metrics import accuracy_score, zero_one_loss

def grid_search(X_train, X_validate, y_train, y_validate, c_range=(0,5),␣
 ↪g_range=(0,5), error_surface=False):

    # Vaš kôd ovdje...
    train_error = []
    validation_error = []

    optimal_c = 0
    optimal_g = 0

    c_start = c_range[0]
    c_end = c_range[1] + 1

    g_start = g_range[0]
    g_end = g_range[1] + 1

    tmp_optimal = float('inf')

    for c in range(c_start, c_end):
        train_row = []
        validation_row = []

        for g in range(g_start, g_end):
            machine = SVC(C = 2**c, gamma = 2**g).fit(X_train, y_train)
            predict_train = machine.predict(X_train)
            predict_validation = machine.predict(X_validate)

            zol_train = zero_one_loss(y_train, predict_train)
            zol_validation = zero_one_loss(y_validate, predict_validation)

            train_row.append(zol_train)
            validation_row.append(zol_validation)

            if zol_validation < tmp_optimal:
                tmp_optimal = zol_validation
                optimal_c = c
```

```
                optimal_g = g

        train_error.append(train_row)
        validation_error.append(validation_row)

    if error_surface:
        return optimal_c, optimal_g, np.matrix(train_error), np.
↪matrix(validation_error)

    return optimal_c, optimal_g
```

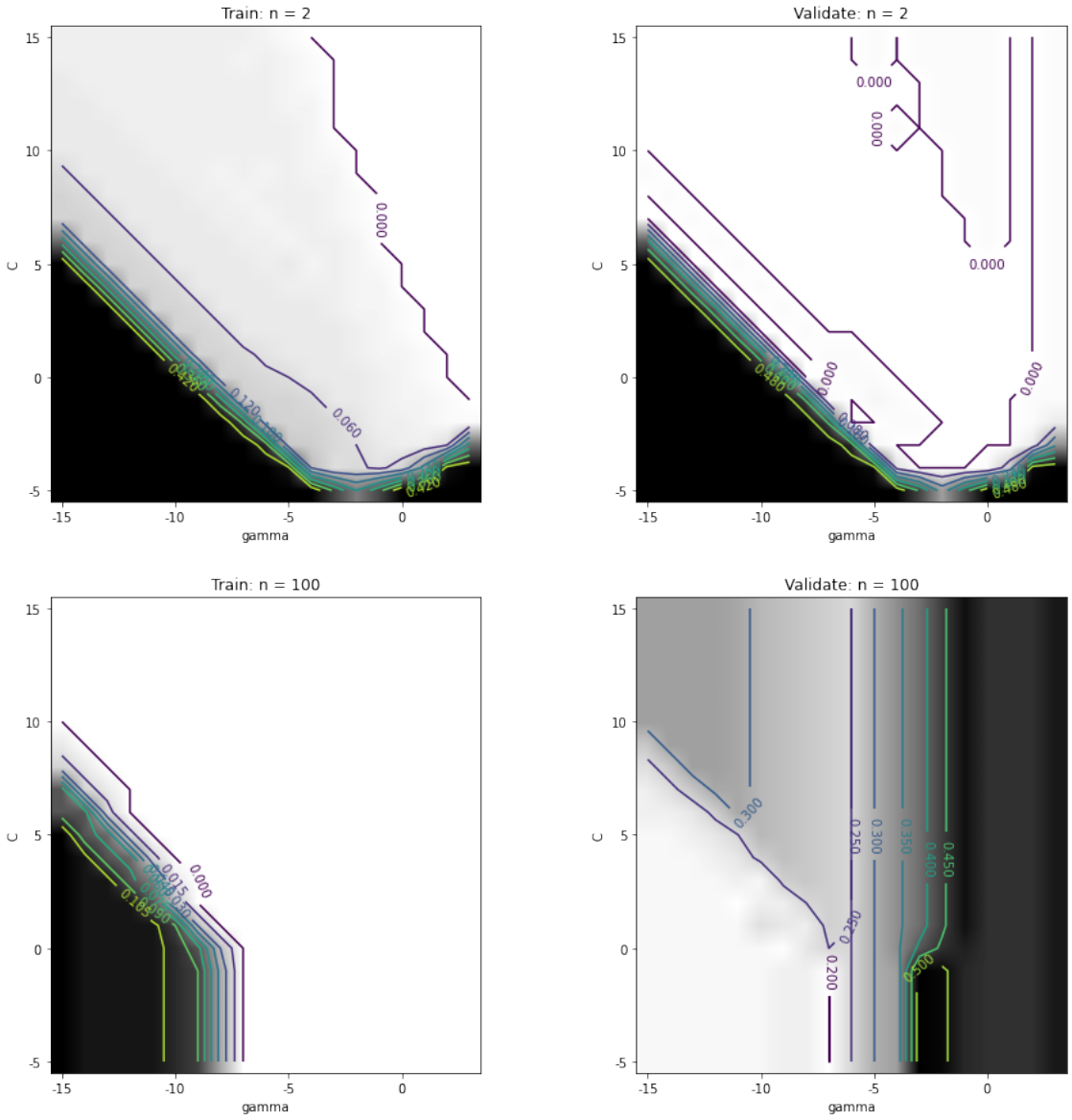
(b) Pomoću funkcije `datasets.make_classification` generirajte **dva** skupa podataka od $N = 200$ primjera: jedan s $n = 2$ dimenzije i drugi s $n = 100$ dimenzija. Primjeri neka dolaze iz dviju klasa, s time da svakoj klasi odgovaraju dvije grupe (`n_clusters_per_class=2`), kako bi problem bio nešto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

Na oba skupa optimirajte SVM s jezgrenom funkcijom RBF, u rešetci $C \in \{2^{-5}, 2^{-4}, \ldots, 2^{15}\}$ i $\gamma \in \{2^{-15}, 2^{-14}, \ldots, 2^{3}\}$. Prikažite površinu pogreške modela na skupu za učenje i skupu za provjeru, i to na oba skupa podataka (ukupno četiri grafikona) te ispišite optimalne kombinacije hiperparametara. Za prikaz površine pogreške modela možete koristiti funkciju `mlutils.plot_error_surface`.

[10]:
```
from sklearn.model_selection import train_test_split

X_1, y_1 = make_classification(n_samples = 200, n_features = 2, n_informative =␣
↪2,
                               n_redundant = 0, n_classes = 2,␣
↪n_clusters_per_class = 2)

X_1_train, X_1_validate, y_1_train, y_1_validate = train_test_split(X_1, y_1,␣
↪test_size = 0.5)



X_2, y_2 = make_classification(n_samples = 200, n_features = 100, n_informative␣
↪= 2,
                               n_redundant = 0, n_classes = 2,␣
↪n_clusters_per_class = 2)

X_2_train, X_2_validate, y_2_train, y_2_validate = train_test_split(X_2, y_2,␣
↪test_size = 0.5)

c_range = (-5, 15)
g_range = (-15, 3)
```

```
optimal_c1, optimal_g1, train_error1, validation_error1 = grid_search(X_train =␣
↪X_1_train, X_validate = X_1_validate,
                                                                      y_train =␣
↪y_1_train, y_validate = y_1_validate,
                                                                      c_range =␣
↪c_range, g_range = g_range, error_surface=True)

optimal_c2, optimal_g2, train_error2, validation_error2 = grid_search(X_train =␣
↪X_2_train, X_validate = X_2_validate,
                                                                      y_train =␣
↪y_2_train, y_validate = y_2_validate,
                                                                      c_range =␣
↪c_range, g_range = g_range, error_surface=True)

print("Dataset 1")
print("Optimal C: " + str(optimal_c1))
print("Optimal G: " + str(optimal_g1))
print()
print("Dataset 2")
print("Optimal C: " + str(optimal_c2))
print("Optimal G: " + str(optimal_g2))

figure(figsize(15, 15))

subplot(2, 2, 1)
title(f'Train: n = {2}')
plot_error_surface(train_error1, c_range, g_range)

subplot(2, 2, 2)
title(f'Validate: n = {2}')
plot_error_surface(validation_error1, c_range, g_range)

subplot(2, 2, 3)
title(f'Train: n = {100}')
plot_error_surface(train_error2, c_range, g_range)

subplot(2, 2, 4)
title(f'Validate: n = {100}')
plot_error_surface(validation_error2, c_range, g_range)
```

```
Dataset 1
Optimal C: -4
Optimal G: -3

Dataset 2
Optimal C: -5
Optimal G: -7
```



Q: Razlikuje li se površina pogreške na skupu za učenje i skupu za ispitivanje? Zašto?
Q: U prikazu površine pogreške, koji dio površine odgovara prenaučenosti, a koji podnaučenosti? Zašto?
Q: Kako broj dimenzija n utječe na površinu pogreške, odnosno na optimalne hiperparametre (C^*, γ^*)?
Q: Preporuka je da povećanje vrijednosti za γ treba biti popraćeno smanjenjem vrijednosti za C. Govore li vaši rezultati u prilog toj preporuci? Obrazložite.

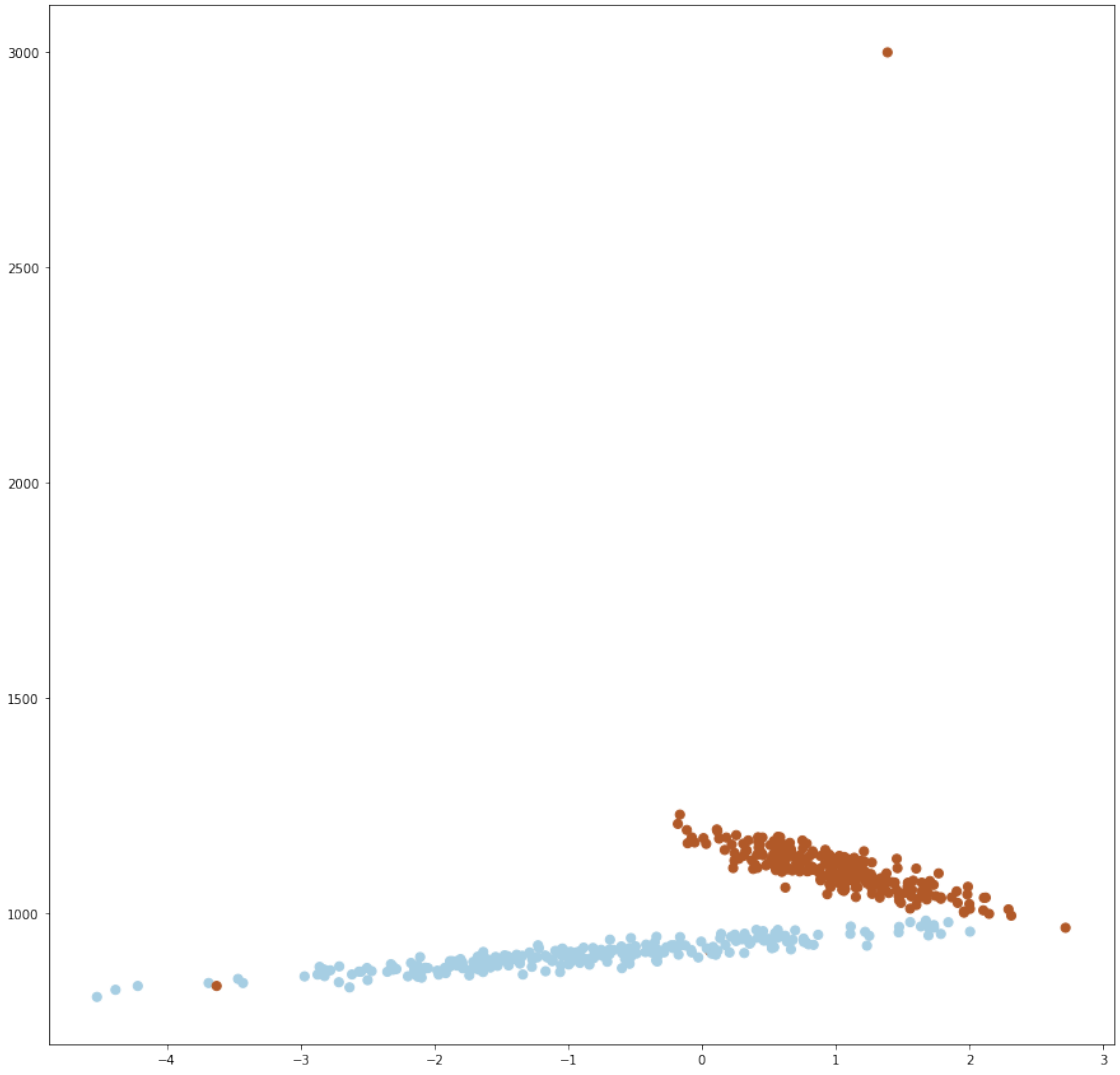
0.1.6 4. Utjecaj standardizacije značajki kod SVM-a

U prvoj laboratorijskoj vježbi smo pokazali kako značajke različitih skala mogu onemogućiti interpretaciju naučenog modela linearne regresije. Međutim, ovaj problem javlja se kod mnogih modela

pa je tako skoro uvijek bitno prije treniranja skalirati značajke, kako bi se spriječilo da značajke s većim numeričkim rasponima dominiraju nad onima s manjim numeričkim rasponima. To vrijedi i za SVM, kod kojega skaliranje nerijetko može znatno poboljšati rezultate. Svrha ovog zadataka jest eksperimentalno utvrditi utjecaj skaliranja značajki na točnost SVM-a.

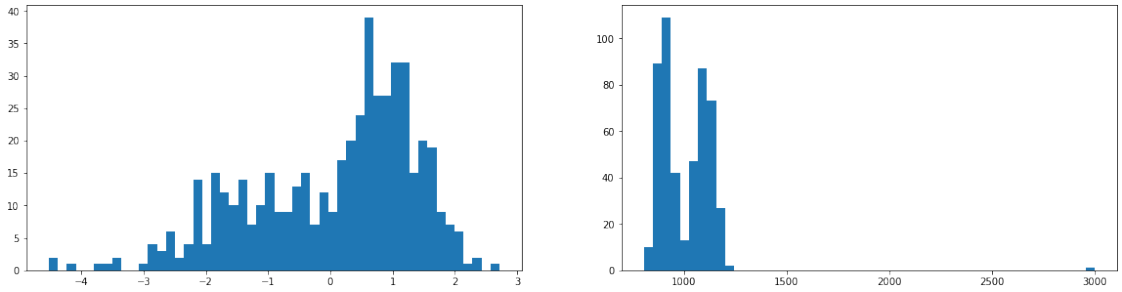
Generirat ćemo dvoklasni skup od $N = 500$ primjera s $n = 2$ značajke, tako da je dimenzija x_1 većeg iznosa i većeg raspona od dimenzije x_0, te ćemo dodati jedan primjer koji vrijednošću značajke x_1 odskače od ostalih primjera:

[11]:
```
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

plot_2d_svc_problem(X, y)
```



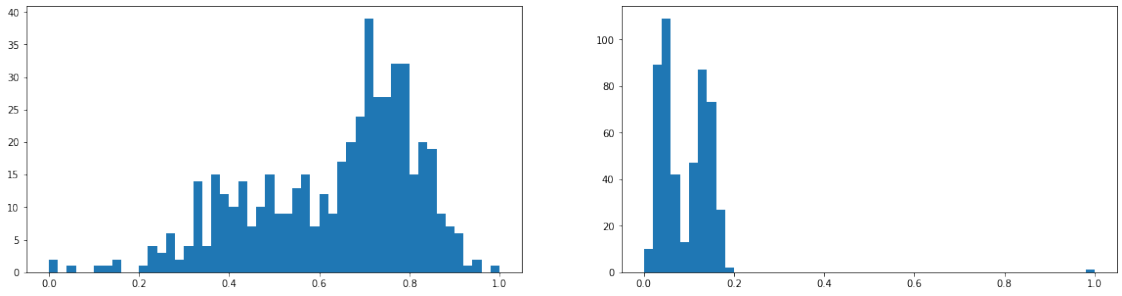
(a) Proučite funkciju za iscrtavanje histograma `hist`. Prikažite histograme vrijednosti značajki x_0 i x_1 (ovdje i u sljedećim zadatcima koristite `bins=50`).

[12]:
```
# Vaš kôd ovdje...
figure(figsize(20, 5))

subplot(1, 2, 1)
hist(X[:,0], bins = 50);

subplot(1, 2, 2)
hist(X[:,1], bins = 50);
```



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

[13]:
```
from sklearn.preprocessing import MinMaxScaler

# Vaš kôd ovdje...
min_max_scaled = MinMaxScaler().fit_transform(X)

figure(figsize(20, 5))

subplot(1, 2, 1)
hist(min_max_scaled[:,0], bins = 50);

subplot(1, 2, 2)
hist(min_max_scaled[:,1], bins = 50);
```

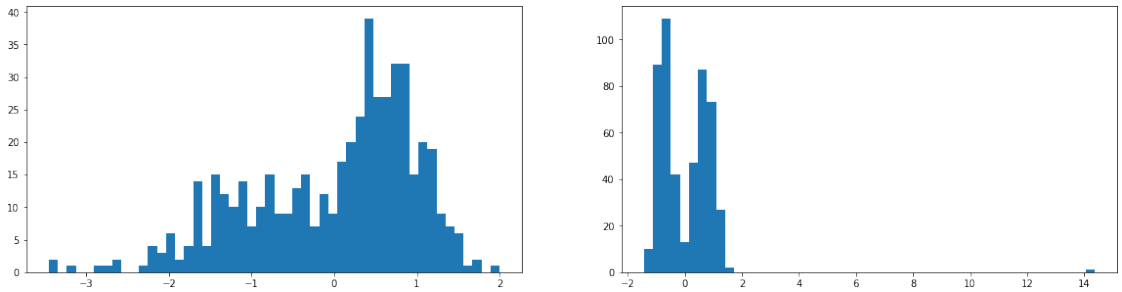


Q: Kako radi ovo skaliranje? **Q:** Dobiveni histogrami su vrlo slični. U čemu je razlika?

(c) Proučite razred `preprocessing.StandardScaler`. Prikažite histograme vrijednosti značajki x_0 i x_1 ako su iste skalirane standardnim skaliranjem (ukupno dva histograma).

[14]:
```
from sklearn.preprocessing import StandardScaler

# Vaš kôd ovdje...
```

```
standard_scaled = StandardScaler().fit_transform(X)

figure(figsize(20, 5))

subplot(1, 2, 1)
hist(standard_scaled[:,0], bins = 50);

subplot(1, 2, 2)
hist(standard_scaled[:,1], bins = 50);
```



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

(d) Podijelite skup primjera na skup za učenje i skup za ispitivanje u omjeru 1:1. Trenirajte SVM s jezgrenom funkcijom RBF na skupu za učenje i ispitajte točnost modela na skupu za ispitivanje, koristeći tri varijante gornjeg skupa: neskalirane značajke, standardizirane značajke i min-max skaliranje. Koristite podrazumijevane vrijednosti za C i γ. Izmjerite točnost svakog od triju modela na skupu za učenje i skupu za ispitivanje. Ponovite postupak više puta (npr. 30) te uprosječite rezultate (u svakom ponavljanju generirajte podatke kao što je dano na početku ovog zadatka).

NB: Na skupu za učenje treba najprije izračunati 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 učenje (funkcija `transform`).

[15]:
```
def generate():
    X, y = make_classification(n_samples = 500, n_features = 2, n_classes = 2,␣
 ↪n_redundant = 0, n_clusters_per_class = 1)
    X[:,1] = X[:,1]*100+1000
    X[0,1] = 3000

    return X, y
```

[16]:
```
def split_generate(test_size = 0.5):
    X, y = generate()
    to_return = train_test_split(X, y, test_size = test_size)
    return to_return
```

[17]:
```
def scale_train(X, type='unscaled'):
    if type == 'standard' or type == 's':
        standardized = StandardScaler().fit_transform(X)
        return standardized
    elif type == 'min-max' or type == 'mm':
        min_maxed = MinMaxScaler().fit_transform(X)
        return min_maxed
    return X
```

[18]:
```
def scale_validate(X, type='unscaled'):
    if type == 'standard' or type == 's':
        standardized = StandardScaler().transform(X)
        return standardized
    elif type == 'min-max' or type == 'mm':
        min_maxed = MinMaxScaler().transform(X)
        return min_maxed
    return X
```

[19]:
```
def compute_accuracy(model, X_train, X_validate, y_train, y_validate):
    train_prediction = model.fit(X_train, y_train).predict(X_train)
    validate_prediction = model.fit(X_validate, y_validate).predict(X_validate)

    accuracy_train = accuracy_score(y_train, train_prediction)
    accuracy_validate = accuracy_score(y_validate, validate_prediction)

    return accuracy_train, accuracy_validate
```

[20]:
```
def scale_generate(X, type='unscaled', validation_type='train'):
    if validation_type == 'validate' or validation_type == 'test':
        return scale_validate(X, type)
    return scale_train(X, type)
```

[21]:
```
# Vaš kôd ovdje...

num = 30

machine = SVC()

total_unscaled_train_accuracy = 0
total_unscaled_validate_accuracy = 0

total_standardized_train_accuracy = 0
total_standardized_validate_accuracy = 0

total_minmax_train_accuracy = 0
total_minmax_validate_accuracy = 0
```

```
# for i in range(num):
#     X_train, X_validate, y_train, y_validate = split_generate(69, 0.5)

#     X_train_unscaled = scale_generate(X_train, type='unscaled',␣
 ↪validation_type='train')
#     X_validate_unscaled = scale_generate(X_validate, type='unscaled',␣
 ↪validation_type='validate')
#     acc_unscaled_train, acc_unscaled_validate = compute_accuracy(machine,␣
 ↪X_train, X_validate, y_train, y_validate)

#     X_train_standard = scale_generate(X_train, type='standard',␣
 ↪validation_type='train')
#     X_validate_standard = scale_generate(X_validate, type='standard',␣
 ↪validation_type='validate')
#     acc_std_train, acc_std_validate = compute_accuracy(machine,␣
 ↪X_train_standard, X_validate_standard, y_train, y_validate)

#     X_train_minmax = scale_generate(X_train, type='min-max',␣
 ↪validation_type='train')
#     X_validate_minmax = scale_generate(X_validate, type='min-max',␣
 ↪validation_type='validate')
#     acc_minmax_train, acc_minmax_validate = compute_accuracy(machine,␣
 ↪X_train_minmax, X_validate_minmax, y_train, y_validate)

#     total_unscaled_train_accuracy += acc_unscaled_train
#     total_unscaled_validate_accuracy += acc_unscaled_validate

#     total_standardized_train_accuracy += acc_std_train
#     total_standardized_validate_accuracy += acc_std_validate

#     total_minmax_train_accuracy += acc_minmax_train
#     total_minmax_validate_accuracy += acc_minmax_validate
for i in range(num):
    X_train, X_validate, y_train, y_validate = split_generate()

    machine.fit(X_train, y_train)
    train_p = machine.predict(X_train)
    validate_p = machine.predict(X_validate)

    total_unscaled_train_accuracy += accuracy_score(y_train, train_p)
    total_unscaled_validate_accuracy += accuracy_score(y_validate, validate_p)

    std_scaler = StandardScaler()
    X_std_t = std_scaler.fit_transform(X_train)
    X_std_v = std_scaler.transform(X_validate)
    machine.fit(X_std_t, y_train)
```

```
    std_t_p = machine.predict(X_std_t)
    std_v_p = machine.predict(X_std_v)

    total_standardized_train_accuracy += accuracy_score(y_train, std_t_p)
    total_standardized_validate_accuracy += accuracy_score(y_validate, std_v_p)

    mm_scaler = MinMaxScaler()
    X_mm_t = mm_scaler.fit_transform(X_train)
    X_mm_v = mm_scaler.transform(X_validate)
    machine.fit(X_mm_t, y_train)
    mm_t_p = machine.predict(X_mm_t)
    mm_v_p = machine.predict(X_mm_v)

    total_minmax_train_accuracy += accuracy_score(y_train, mm_t_p)
    total_minmax_validate_accuracy += accuracy_score(y_validate, mm_v_p)



total_unscaled_train_accuracy /= num
total_unscaled_validate_accuracy /= num

total_standardized_train_accuracy /= num
total_standardized_validate_accuracy /= num

total_minmax_train_accuracy /= num
total_minmax_validate_accuracy /= num

print('UNSCALED accuracy:')
print('Train: ' + str(total_unscaled_train_accuracy))
print('Validate: ' + str(total_unscaled_validate_accuracy))
print()

print('STANDARDIZED accuracy:')
print('Train: ' + str(total_standardized_train_accuracy))
print('Validate: ' + str(total_standardized_validate_accuracy))
print()

print('MINMAX accuracy:')
print('Train: ' + str(total_minmax_train_accuracy))
print('Validate: ' + str(total_minmax_validate_accuracy))
print()
```

```
UNSCALED accuracy:
Train: 0.7419999999999998
Validate: 0.7222666666666666

STANDARDIZED accuracy:
Train: 0.9606666666666668
```

```
Validate: 0.9533333333333333

MINMAX accuracy:
Train: 0.9549333333333333
Validate: 0.9469333333333333
```

Q: Jesu li rezultati očekivani? Obrazložite. **Q:** Bi li bilo dobro kada bismo funkciju `fit_transform` primijenili na cijelom skupu podataka? Zašto? Bi li bilo dobro kada bismo tu funkciju primijenili zasebno na skupu za učenje i zasebno na skupu za ispitivanje? Zašto?

0.1.7 5. Algoritam k-najbližih susjeda

U ovom zadatku promatrat ćemo jednostavan klasifikacijski model imena **algoritam k-najbližih susjeda**. Najprije ćete ga samostalno isprogramirati kako biste se detaljno upoznali s radom ovog modela, a zatim ćete prijeći na analizu njegovih hiperparametara (koristeći ugrađeni razred, radi efikasnosti).

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

[22]:
```
def euclidean_distance(x1, x2):
    return np.linalg.norm(np.array(x1) - np.array(x2))
```

[23]:
```
def get_neighbours(X_train, y_train, x):
    to_return = [[euclidean_distance(train, x), y] for train, y in zip(X_train,␣
 ↪y_train)]
    return to_return
```

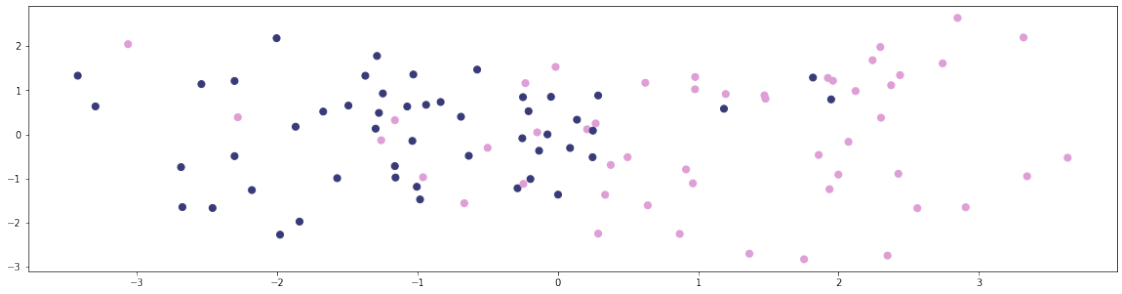
[24]:
```
def get_votes(neighbours, k):
    sorted_k_neighbours = sorted(neighbours)[:k]
    to_return = [prediction for (distance, prediction) in sorted_k_neighbours]
    return to_return
```

[25]:
```
from numpy.linalg import norm

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

    def fit(self, X_train, y_train):
        # Vaš kôd ovdje...
        self.X_train = X_train
        self.y_train = y_train
```

```
    def predict(self, X_test):
        # Vaš kôd ovdje...
        k = self.k
        X_train = self.X_train
        y_train = self.y_train
        predictions = []

        for x in X_test:
            neighbours = get_neighbours(X_train, y_train, x)
            vote = get_votes(neighbours, k)
            predictions.append(max(vote, key=vote.count))

        return predictions
```

(b) Kako biste se uvjerili da je Vaša implementacija ispravna, usporedite ju s onom u razredu `neighbors.KNeighborsClassifier`. Budući 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 uspoređuju polja; `numpy.all`).

[26]:
```
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)
plot_2d_clf_problem(X_art, y_art)
```



[27]:
```
from sklearn.neighbors import KNeighborsClassifier

# Vaš kôd ovdje...
knc = KNeighborsClassifier(n_neighbors = 3)
knc.fit(X_art, y_art)
prediction = knc.predict(X_art)
knn = KNN(n_neighbors = 3)
knn.fit(X_art, y_art)
```

```
my_prediction = np.array(knn.predict(X_art))
print(my_prediction)
print()
print(prediction)

numpy.all(prediction == my_prediction)
```

```
[1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 1 1 0 0
 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 0 1
 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0]

[1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 1 1 0 0
 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 0 1
 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0]
```

[27]: `True`

0.1.8 6. Analiza algoritma k-najbližih susjeda

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

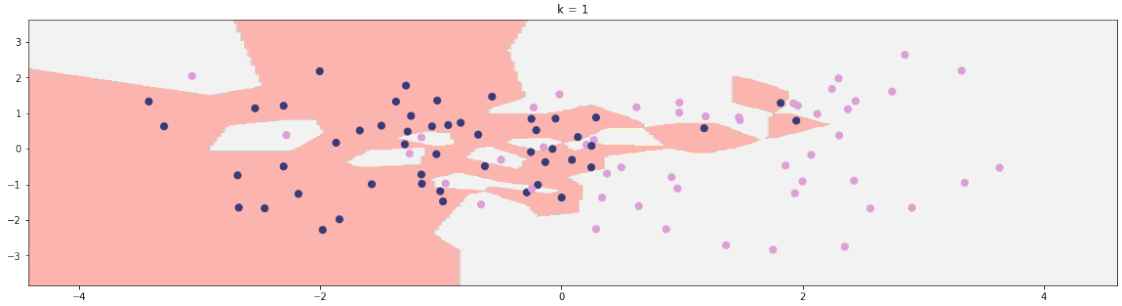
Kako bismo dobili pouzdanije rezultate, potrebno je neke od eksperimenata ponoviti na različitim skupovima podataka i zatim uprosječiti dobivene vrijednosti pogrešaka. Koristite funkciju: `knn_eval` koja trenira i ispituje model k-najbližih susjeda na ukupno `n_instances` primjera, i to tako da za svaku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

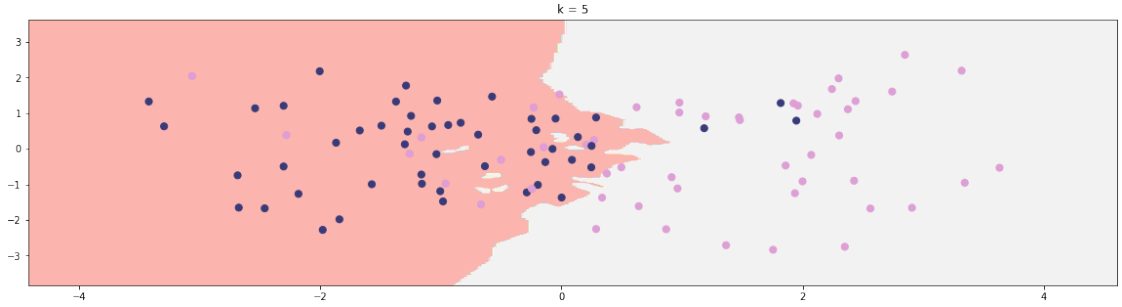
(a) Na podatcima iz zadatka 5, pomoću funkcije `plot_2d_clf_problem` iscrtajte prostor primjera i područja koja odgovaraju prvoj odnosno drugoj klasi. Ponovite ovo za $k \in [1, 5, 20, 100]$.

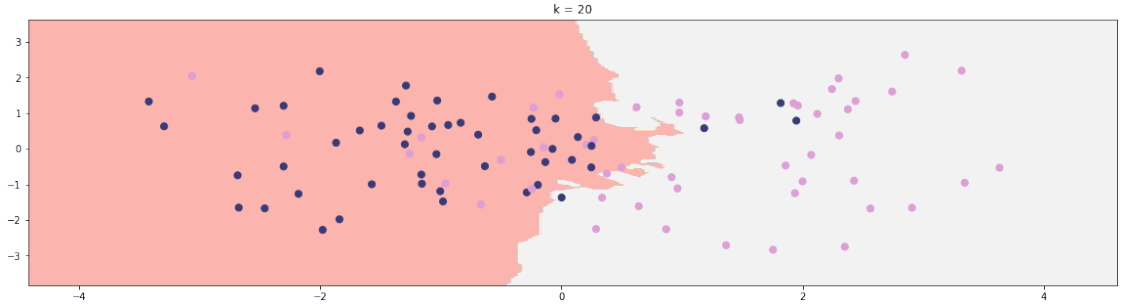
NB: Implementacija algoritma `KNeighborsClassifier` iz paketa `scikit-learn` vjerojatno će raditi brže od Vaše implementacije, pa u preostalim eksperimentima koristite nju.

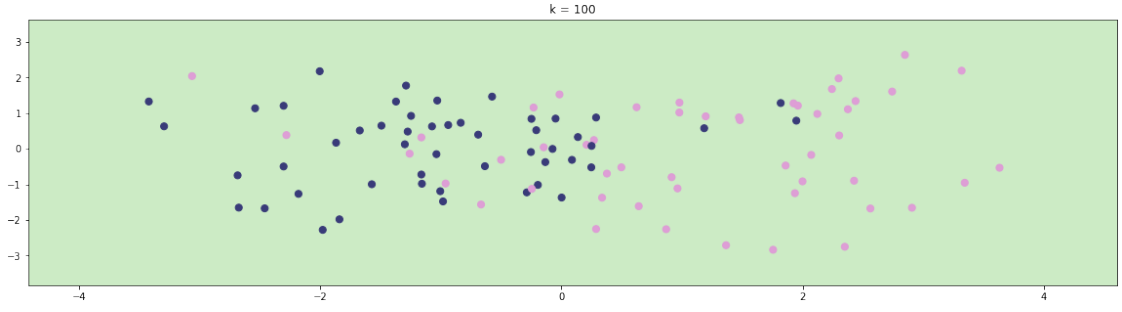
[28]:
```
# Vaš kôd ovdje...
ks = [1, 5, 20, 100]

for k in ks:
    knc = KNeighborsClassifier(n_neighbors = k)
    knc.fit(X_art, y_art)
    plt.figure()
```

```
plot_2d_clf_problem(X_art, y_art, knc.predict)
title('k = ' + str(k))
```









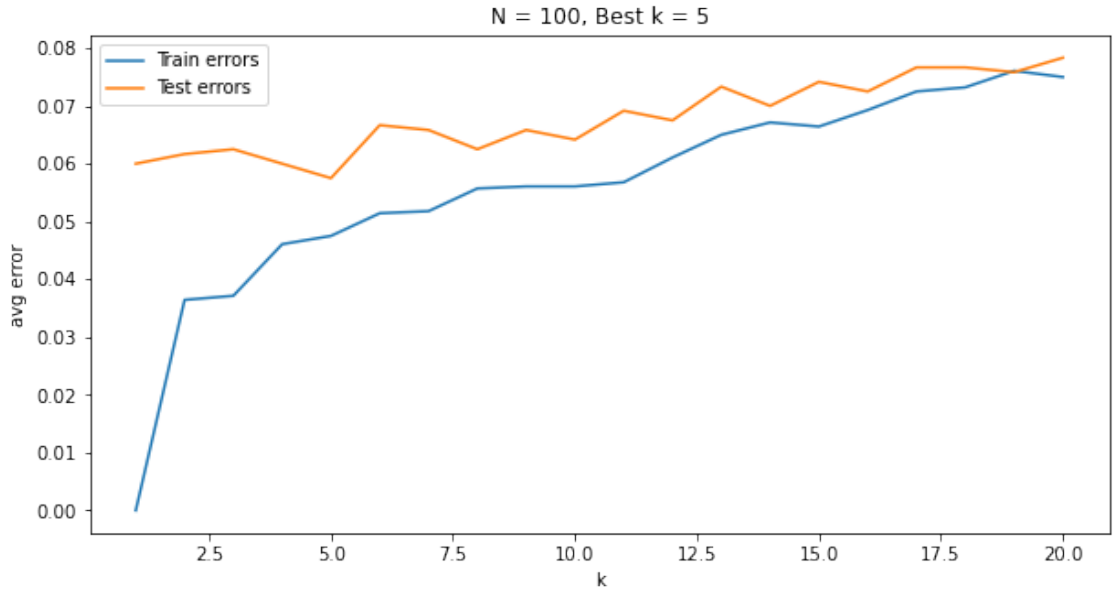
Q: Kako k utječe na izgled granice između klasa?
Q: Kako se algoritam ponaša u ekstremnim situacijama: $k = 1$ i $k = 100$?

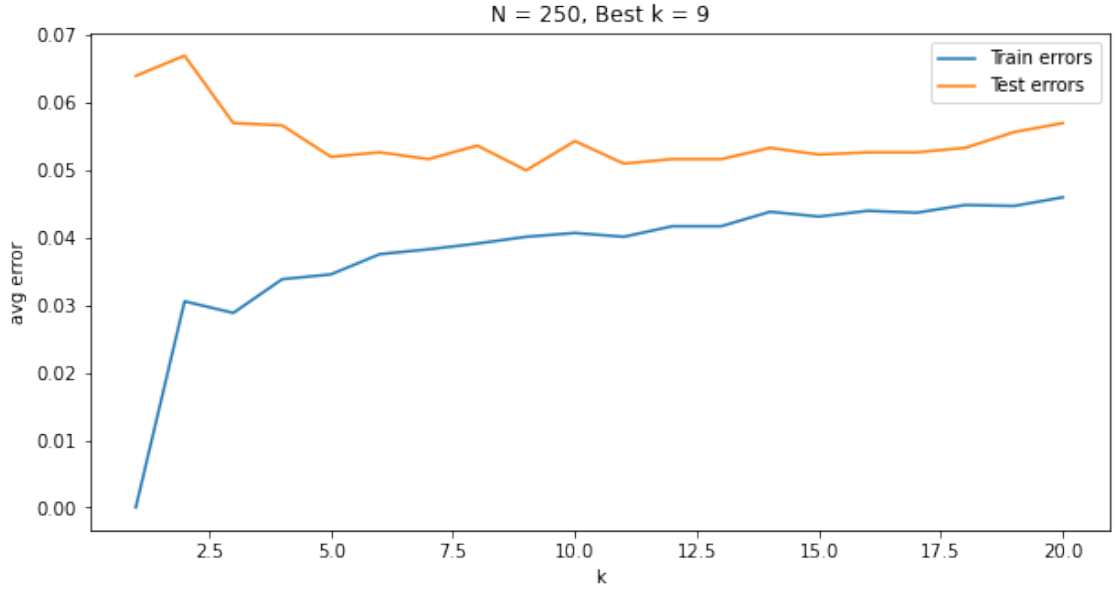
(b) Pomoću funkcije `knn_eval`, iscrtajte pogreške učenja i ispitivanja kao funkcije hiperparametra $k \in \{1, \ldots, 20\}$, za $N = \{100, 250, 750\}$ primjera. Načinite 3 zasebna grafikona. Za svaki ispišite optimalnu vrijednost hiperparametra k (najlakše kao naslov grafikona; vidi `plt.title`).

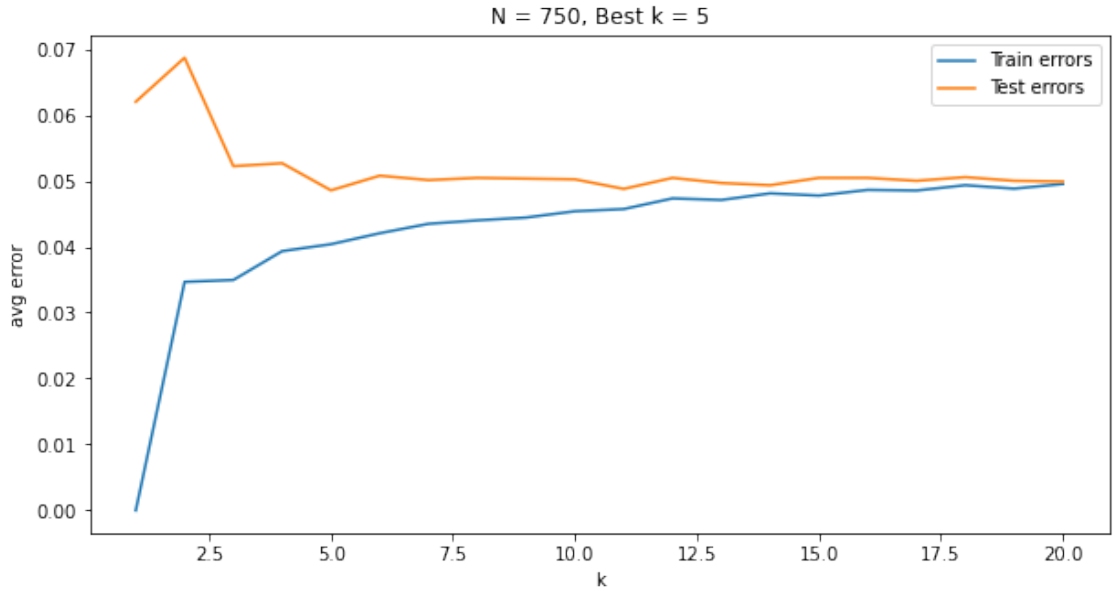
[29]:
```
# Vaš kôd ovdje...

k_range = (1, 20)
N_instances = [100, 250, 750]
plt.tight_layout()
figure(figsize(10, 5))
for N in N_instances:

    ks, best_k, train_errors, test_errors = knn_eval(n_instances=N,␣
 ↪n_features=2, n_classes=2, n_informative=2,
                  test_size=0.3, k_range=k_range, n_trials=40)
    plt.figure()
    title(f'N = {N}, Best k = {best_k}')
    plot(ks, train_errors, label="Train errors")
    plot(ks, test_errors, label='Test errors')
    plt.xlabel('k')
    plt.ylabel('avg error')
    plt.legend()
```

```
<Figure size 1440x360 with 0 Axes>
```

```
<Figure size 720x360 with 0 Axes>
```


N = 100, Best k = 5
Train errors
Test errors
avg error
k
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


N = 250, Best k = 9
Train errors
Test errors
avg error
k
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Q: Kako se mijenja optimalna vrijednost hiperparametra k s obzirom na broj primjera N? Zašto?
Q: Kojem području odgovara prenaučenost, a kojem podnaučenost modela? Zašto?
Q: Je li uvijek moguće doseći pogrešku od 0 na skupu za učenje?

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

Koristite funkciju `mlutils.knn_eval` na dva načina. U oba koristite $N = 1000$ primjera, $n = 10$ značajki i $K = 5$ klasa, ali za prvi neka su svih 10 značajki bitne, a za drugi neka je bitno samo 5 od 10 značajki. Ispišite pogreške učenja i ispitivanja za oba modela za optimalnu vrijednost k (vrijednost za koju je ispitna pogreška najmanja).

[30]:
```
# Vaš kôd ovdje...
N = 1000
n = 10
K = 5
n_informative = [10, 5]

for n_inf in n_informative:
    ks, best_k, train_errors, test_errors = knn_eval(n_instances=N,␣
 ↪n_features=n, n_classes=K, n_informative=n_inf,
                 test_size=0.3, n_trials=40)
    print(f'n_informative = {n_inf}, best_k = {best_k}')
```

```
    print(f'train error = {train_errors[best_k - 1]}')
    print(f'test error = {test_errors[best_k - 1]}')
    print()
```

```
n_informative = 10, best_k = 7
train error = 0.08824999999999997
test error = 0.12416666666666668

n_informative = 5, best_k = 12
train error = 0.16621428571428568
test error = 0.20516666666666666
```

Q: Je li algoritam k-najbližih susjeda osjetljiv na nebitne značajke? Zašto?
Q: Je li ovaj problem izražen i kod ostalih modela koje smo dosad radili (npr. logistička regresija)?
Q: Kako bi se model k-najbližih susjeda ponašao na skupu podataka sa značajkama različitih skala? Detaljno pojasnite.

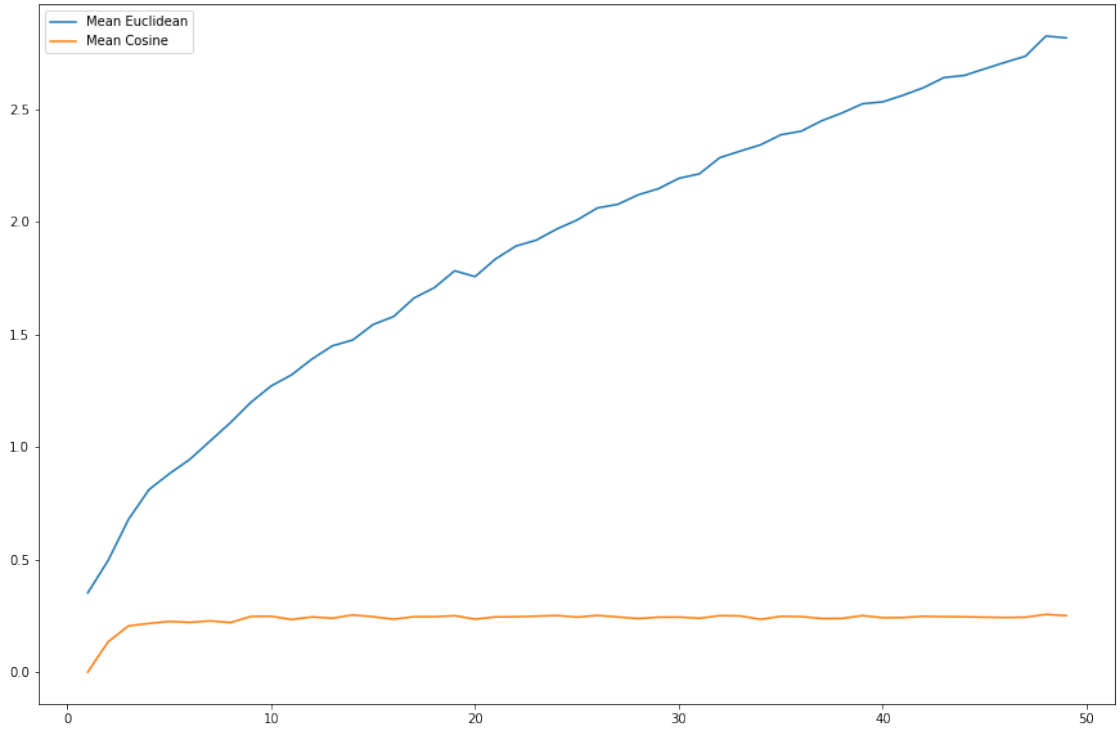
0.1.9 7. "Prokletstvo dimenzionalnosti"

"Prokletstvo dimenzionalnosti" zbirni je naziv za niz fenomena povezanih s visokodimenzijskim prostorima. Ti fenomeni, koji se uglavnom protive našoj intuiciji, u većini slučajeva dovode do toga da se s porastom broja dimenzija (značajki) smanjenje točnost modela.

Općenito, povećanje dimenzija dovodi do toga da sve točke u ulaznome prostoru postaju (u smislu euklidske udaljenosti) sve udaljenije jedne od drugih te se, posljedično, gube razlike u udaljenostima između točaka. Eksperimentalno ćemo provjeriti da je to doista slučaj. Proučite funkciju `metrics.pairwise_distances`. Generirajte 100 slučajnih vektora u različitim dimenzijama $n \in [1, 2, \ldots, 50]$ dimenzija te izračunajte *prosječnu* euklidsku udaljenost između svih parova tih vektora. Za generiranje slučajnih vektora koristite funkciju `numpy.random.random`. Na istom grafu skicirajte i krivulju za prosječne kosinusne udaljenosti (parametar `metric`).

[31]:
```
from sklearn.metrics.pairwise import pairwise_distances
from math import cos

# Vaš kôd ovdje...
dimensions = [i for i in range(1, 50)]
num = [i for i in range(100)]

euclidean = []
cosine = []

figure(figsize(15, 10))

for n in dimensions:
    pairwise_euclidean = []
    pairwise_cosine = []
    vector = [np.random.random(size = n) for i in num]
```

```
    pairwise_euclidean = pairwise_distances(vector, metric = 'euclidean')
    pairwise_cosine = pairwise_distances(vector, metric = 'cosine')

    euclidean.append(mean(pairwise_euclidean))
    cosine.append(mean(pairwise_cosine))

plot(dimensions, euclidean, label='Mean Euclidean')
plot(dimensions, cosine, label='Mean Cosine')
plot.xlabel = 'dimension'
plot.ylabel = 'pairwise distance'
legend()
show()
```



Q: Pokušajte objasniti razlike u rezultatima. Koju biste od ovih dviju mjera koristili za klasifikaciju visokodimenzijskih podataka?
Q: Zašto je ovaj problem osobito izražen kod algoritma k-najbližih susjeda?