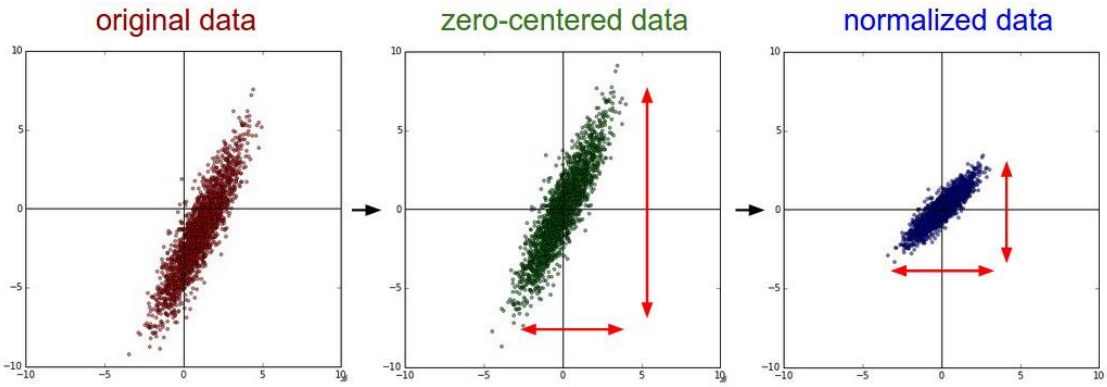
Normalizarea datelor. Mașini cu vectori suport (SVM)

1. Normalizarea datelor



Metode obișnuite de preprocesare a datelor. În partea stângă sunt reprezentate datele 2D originale. În mijloc acestea sunt centrate în 0, prin scăderea mediei pe fiecare dimensiune. În partea dreaptă fiecare dimensiune este scalată folosind deviația standard corespunzătoare Spre deosebire de imaginea din centru, unde datele au lungimi diferite pe cele două axe, aici ele sunt egale..

1.1. Standardizarea

- transformă vectorii de caracteristici astfel încât fiecare să aibă medie 0 și deviație standard 1

$x\_scaled = \frac{x - mean(x)}{\sigma}$, unde x_mean- media valorilor lui x

σ - deviația standard

```
from sklearn import preprocessing
import numpy as np

x_train = np.array([[1, -1, 2], [2, 0, 0], [0, 1, -1]], dtype=np.float64)
x_test = np.array([[-1, 1, 0]], dtype=np.float64)

# facem statisticile pe datele de antrenare
scaler = preprocessing.StandardScaler()
scaler.fit(x_train)

# afisam media
print(scaler.mean_)                 # => [1.  0.  0.33333333]
# afisam deviatia standard
print(scaler.scale_)                # => [0.81649658 0.81649658 1.24721913]

# scalam datele de antrenare
scaled_x_train = scaler.transform(x_train)
print(scaled_x_train)        # => [[0.          -1.22474487  1.33630621]
                             #     [1.22474487   0.          -0.26726124]
                             #     [-1.22474487  1.22474487  -1.06904497]]

# scalam datele de test
scaled_x_test = scaler.transform(x_test)
print(scaled_x_test)        # => [[-2.44948974  1.22474487 -0.26726124]]
```

1.2. Scalarea într-un anumit interval

- transformă datele astfel încât valorile fiecărei caracteristici să se încadreze într-un anumit interval, de obicei [0, 1], sau astfel încât valoarea maximă să devină 1
- formula generală pentru intervalul dat [min_val, max_val]:

$$x\_std = \frac{x - min(x)}{max(x) - min(x)}$$

$$x\_scaled = x\_std * (max\_val - min\_val) + min\_val$$

```
from sklearn import preprocessing
import numpy as np

x_train = np.array([[1, -1, 2], [2, 0, 0], [0, 1, -1]], dtype=np.float64)
x_test = np.array([[-1, 1, 0]], dtype=np.float64)

# facem statisticile pe datele de antrenare
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1)) # (0, 1) default
min_max_scaler.fit(x_train)

# scalam datele de antrenare
scaled_x_train = min_max_scaler.transform(x_train)
print(scaled_x_train)        # => [[0.5  0.    1.        ]
                             #     [1.   0.5   0.33333333]
                             #     [0.   1.    0.        ]]

# scalam datele de test
scaled_x_test = min_max_scaler.transform(x_test)
print(scaled_x_test)         # => [[-0.5  1.  0.33333333]]
```

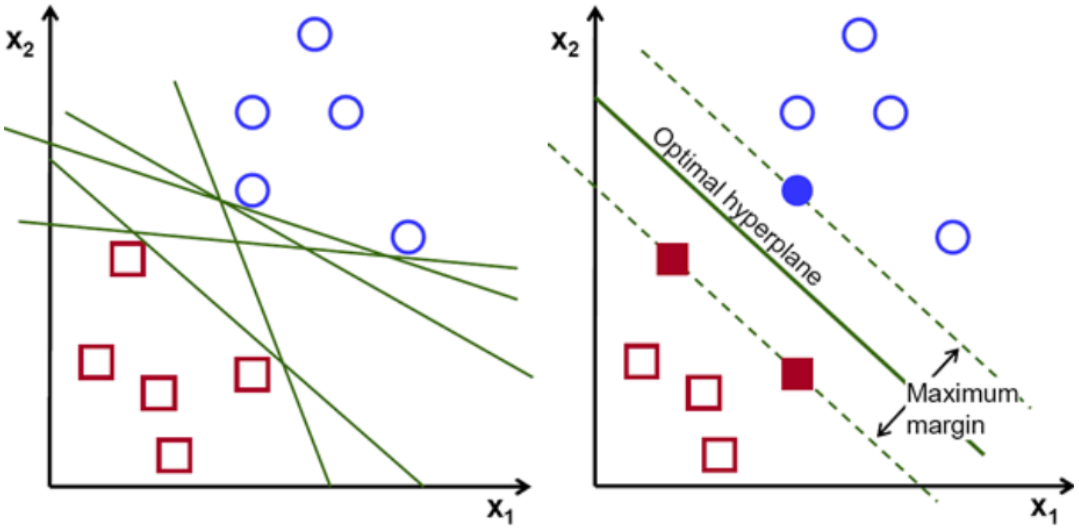
1.3. Normalizarea L1. Normalizarea L2

- scalarea individuală a vectorilor de caracteristici corespunzători fiecărui exemplu astfel încât norma lor să devină 1

Folosind norma L1: $x\_scaled = \frac{X}{||X||_1}, ||X||_1 = \sum_{i=1}^{n} |x_i|$

Folosind norma L2: $x\_scaled = \frac{X}{||X||_2}, ||X||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

2. Mașini cu vectori suport



În partea stângă sunt prezentate drepte de decizie posibile pentru clasificarea celor două tipuri de obiecte. SVM-ul, exemplificat în partea dreaptă, alege hiperplanul care maximizează marginea dintre cele doua clase.

		Funcția de decizie	**Problema de optimizare**
Hard Margin	**Forma primală**	$< x, w > + b \geq 0$	$\min \frac{\|\|w\|\|^2}{2}$ cu constr. $y_i(< x_i, w > + b) - 1 \geq 0$
	Forma duală	$\sum_i \alpha_i y_i < x_i, x > + b \geq 0$	$\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ cu constr. $\alpha_i \geq 0$
Soft Margin	**Forma primală**	$< x, w > + b \geq 0$	$\min \frac{\|\|w\|\|^2}{2} + C \sum_i \xi_i$ cu constr. $y_i(< x_i, w > + b) \geq 1 - \xi_i$
	Forma duală	$\sum_i \alpha_i y_i < x_i, x > + b \geq 0$	$\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ cu constr. $0 \leq \alpha_i \leq C$

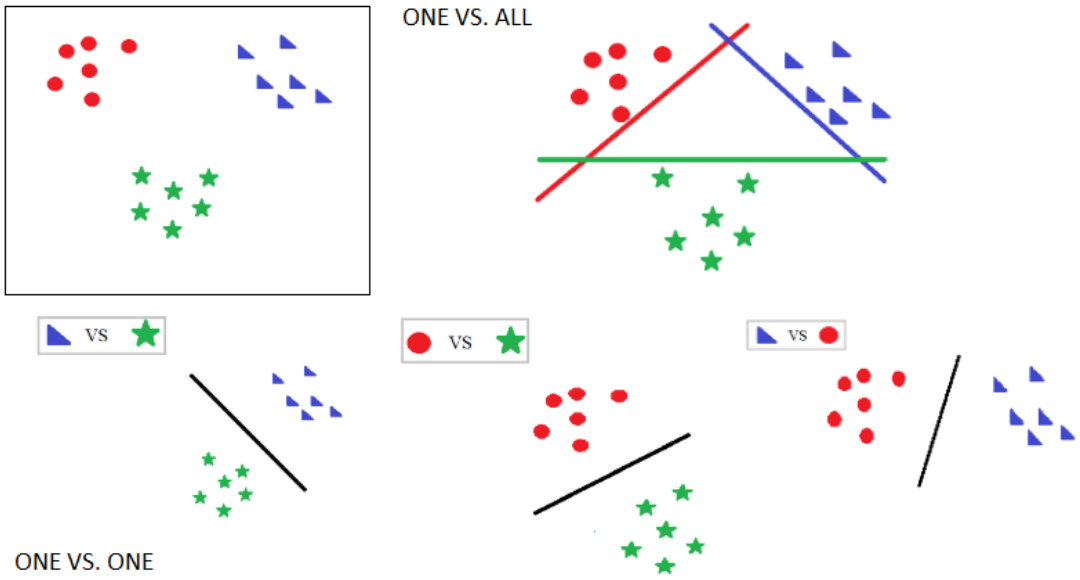
Funcțiile de decizie și problemele de optimizare asociate pentru formele SVM: primală, respectiv duală și folosind soft, respectiv hard margin.

Pentru implementarea acestui algoritm vom folosi biblioteca ***ScikitLearn***. Aceasta este dezvoltată în Python, fiind integrată cu NumPy și pune la dispoziție o serie de algoritmi optimizați pentru probleme de clasificare, regresie și clusterizare.

Instalarea bibliotecii se face prin comanda sistem: `pip install -U scikit-learn`
Importarea modelului:

```
from sklearn import svm
```

Detalii de implementare:



Există două abordări pentru a clasifica datele aparținând mai multor clase:

1. **ONE VS ALL**: Sunt antrenați $num\_classes$ clasificatori, câte unul corespunzător fiecărei clase, care să o diferențieze pe aceasta de toate celelalte (toate celelalte exemple sunt privite ca aparținând aceleiași clase). Eticheta finală pentru un exemplu nou va fi dată de clasificatorul care a obținut scorul maxim.
2. **ONE VS ONE**: Sunt antrenați $\frac{num\_classes * (num\_classes - 1)}{2}$ clasificatori, câte unul corespunzător fiecarei perechi de câte două clase. Eticheta finală pentru un exemplu nou va fi cea care obține cele mai multe voturi pe baza acestor clasificatori.

➔ Implementarea din ScikitLearn are o abordare one-vs-one, adică pentru fiecare 2 clase este antrenat un clasificator binar care să diferențieze între acestea. Astfel, dacă avem un număr de clase egal cu $num\_classes$, vor fi antrenați $\frac{num\_classes * (num\_classes - 1)}{2}$ clasificatori.

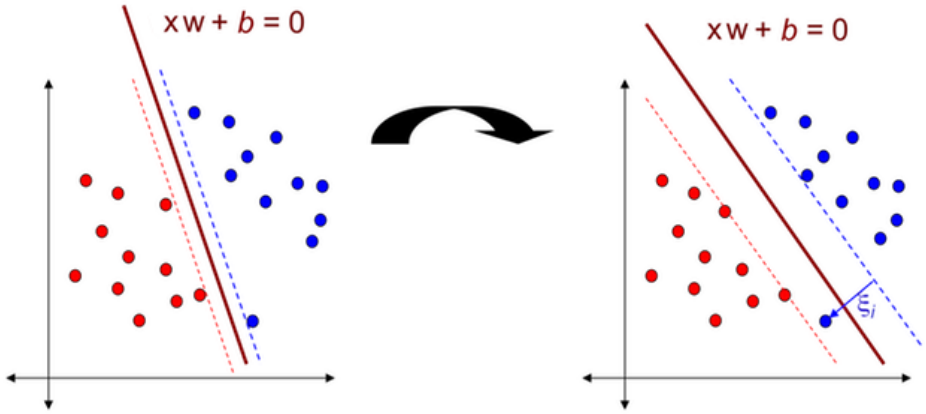
➔ La testare, clasa asignată fiecărui exemplu este cea care obține cele mai multe voturi pe baza acestor clasificatori.

1. Definirea modelului:

```
class sklearn.svm.SVC(C, kernel, gamma)
```

Parametri:

C (float, default = 1.0)



Influența parametrului C în alegerea marginii optime: în partea stângă este folosită abordarea hard margin, în care clasificatorul nu este dispus să clasifice greșit date de antrenare, iar în partea dreaptă este folosită abordarea soft margin. Variabila ξ_i sugerează cât de mult exemplul x_i are voie să depășească marginea.

$$\xi_i = max(0, 1 - y_i(< x, w > + b))$$

- parametru de penalitate pentru eroare, sugerează cât de mult este dispus modelul să evite clasificarea greșită a exemplelor din setul de antrenare:
  - C mare - va fi ales un hiperplan cu o margine mai mică, dacă acesta are rezultate mai bune pe setul de antrenare (mai mulți vectori suport).

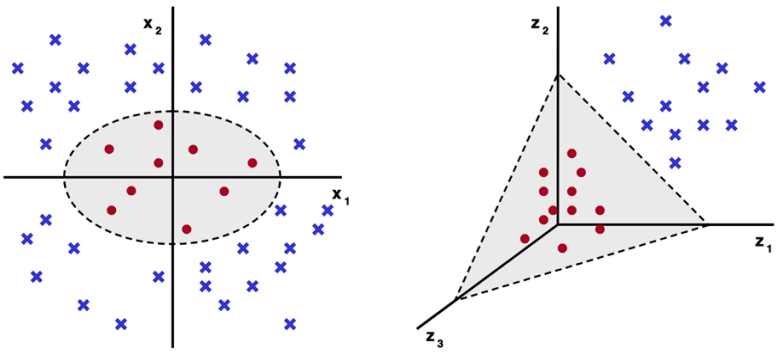
    Dacă C va fi ales prea mare, se poate ajunge la supraînvățare.

  - C mic - va fi ales un hiperplan cu o margine mai mare, chiar dacă acesta duce la clasificarea greșită a unor puncte din setul de antrenare (mai puțini vectori suport).

    Dacă C va fi ales prea mic, modelul nu va fi capabil să învețe, ajungându-se la subînvățare.

kernel (string, default = 'linear')

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Funcțiile kernel sunt folosite atunci când datele nu sunt liniar separabile. Acestea funcționează prin următorii doi pași:

1. Datele sunt scufundate într-un spațiu (Hilbert) cu mai multe dimensiuni
2. Relațiile liniare sunt căutate în acest spațiu

- tipul de kernel folosit: în cadrul laboratorului vom lucra cu 'linear' și 'rbf'

Kernel linear:

$$K(u, v) = u^T v$$

Kernel RBF:

$$K(u, v) = exp(-gamma * ||u - v||^2)$$

gamma (float, default = 'auto', având valoarea $\frac{1}{num\_features}$)

- coeficient pentru kernelul 'rbf'
- dacă gamma = 'scale' va fi folosită valoarea $\frac{1}{num\_features * X.std()}$
- în versiunea 0.22 valoarea default 'auto' va fi schimbată cu 'scale'

2. Antrenarea:

```
svm_model.fit(train_data, train_labels)
```

Parametri:

train_data

- setul de antrenare având exemplele stocate pe linii => dimensiune ($num\_samples$ x $num\_features$)

train_labels

- etichetele corespunzătoare fiecărui exemplu de antrenare

3. Predicția:

```
svm_model.predict(test_data)
```

Parametri:

test_data

- setul de test având exemplele stocate pe linii => dimensiune ($num\_test\_samples$ x $num\_features$)

Funcția întoarce un vector cu $num_test_samples$elemente, fiecare reprezentând id-ul clasei prezise.

Exerciții

În continuare vom lucra pe subsetul **MNIST** primit în laboratorul 3.

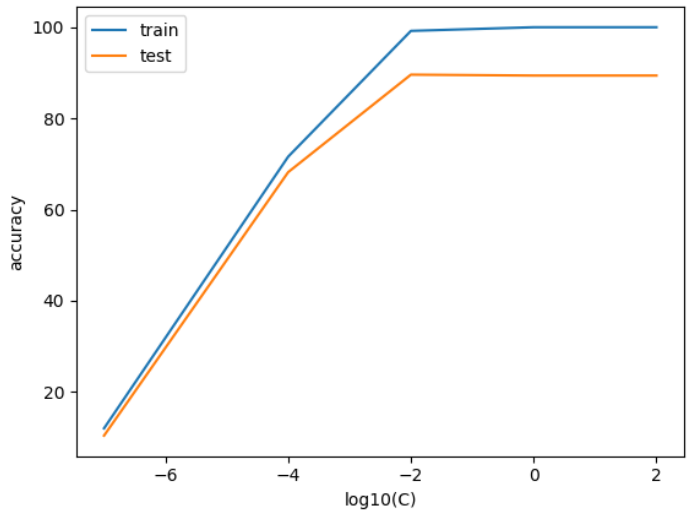
1. Definiți funcția **normalize_data(train_data, test_data, type=None)** care primește ca parametri datele de antrenare, respectiv de testare și tipul de normalizare ({None, *'standard'*, *'min_max'*, *'l1'*, *'l2'*}) și întoarce aceste date normalizate.

2. Definiți funcția **get_accuracy_statistics(train_data, train_labels, test_data, test_labels, Cs, normalization_type=None)** care primește ca parametri datele de antrenare, respectiv de testestare, tipul de normalizare și un vector de valori pentru parametrul C al unui SVM liniar. Aceasta normalizează datele, antrenează câte un SVM pentru fiecare valoare din C și returnează 2 vectori conținând acuratețea fiecărui model pe datele de antrenare, respectiv de test.

   Antrenați un SVM liniar pe subsetul MNIST primit în laboratorul 3, folosind normalizare standard, respectiv l2 și următoarele valori pentru parametrul C: [1e-8, 1e-7, 1e-6, 1].

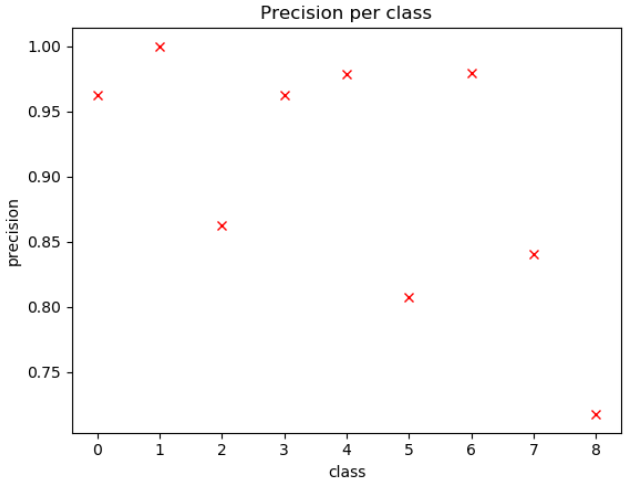
   Afișați acuratețea modelelor atât pe setul de antrenare cât și pe cel de test.

3. Plotați valorile obținute cu modelul antrenat pe datele standardizate în același grafic, folosind pe axa OX $log_{10}(C)$. Cum influențează valoarea lui C acuratețea?



4. Calculați matricea de confuzie pentru cel mai bun model obținut la exercițiul anterior folosind normalizarea L2.

5. Pe baza matricei de confuzie calculați precizia corespunzătoare fiecărei clase și plotați aceste valori.



6. Antrenați un model SVM folosind kernelul ‘rbf’. Încercați atât cu datele normalizate, cât și cu ele neprocesate. Comparați rezultatele cu cele obținute folosind SVM-ul liniar.