Modelul bag-of-words. Normalizarea datelor. Mașini cu vectori suport (SVM).

1. Modelul bag-of-words

➔ este o metodă de reprezentare a datelor de tip text, bazată pe frecvența de apariție a cuvintelor în cadrul documentelor

➔ algoritmul este alcătuit din 2 pași:

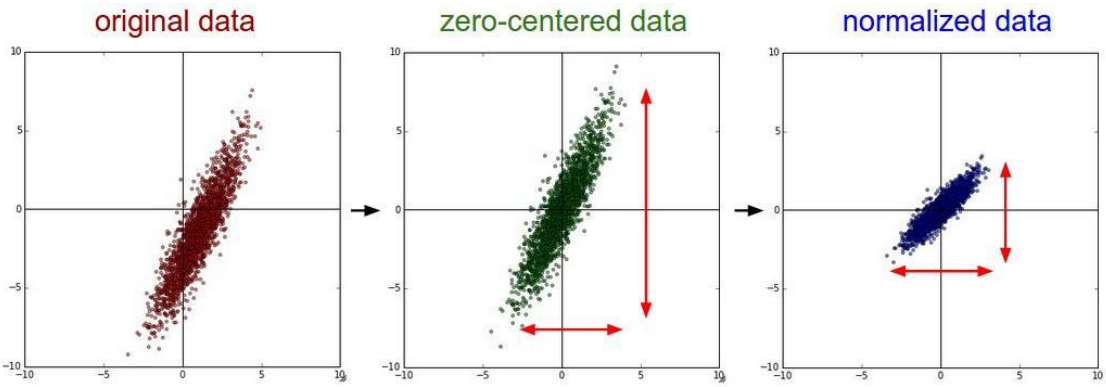
1. definirea unui vocabular prin atribuirea unui id unic fiecărui cuvânt regăsit în setul de date (setul de antrenare)

2. reprezentarea fiecărui document ca un vector de dimensiune egală cu lungimea vocabularului, definit astfel:

$$features[word\_idx] = numărul\ de\ apariţii\ al\ cuvântului\ cu\ id - ul\ word\_idx$$

2. Normalizarea datelor

2.1. Standardizarea



Metode obișnuite de preprocesare a datelor. În partea **stângă** sunt reprezentate datele 2D originale. În **mijloc** acestea sunt centrate în origine, 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.

- 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)
```

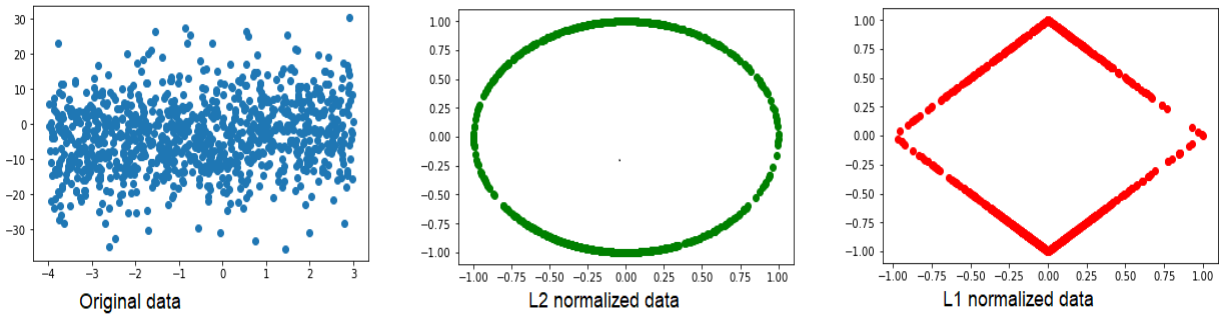
```
# facem statisticile pe datele de antrenare
scaler = preprocessing.StandardScaler()
scaler.fit(x_train)

# 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]]
```

2.2. Normalizarea L1. Normalizarea L2



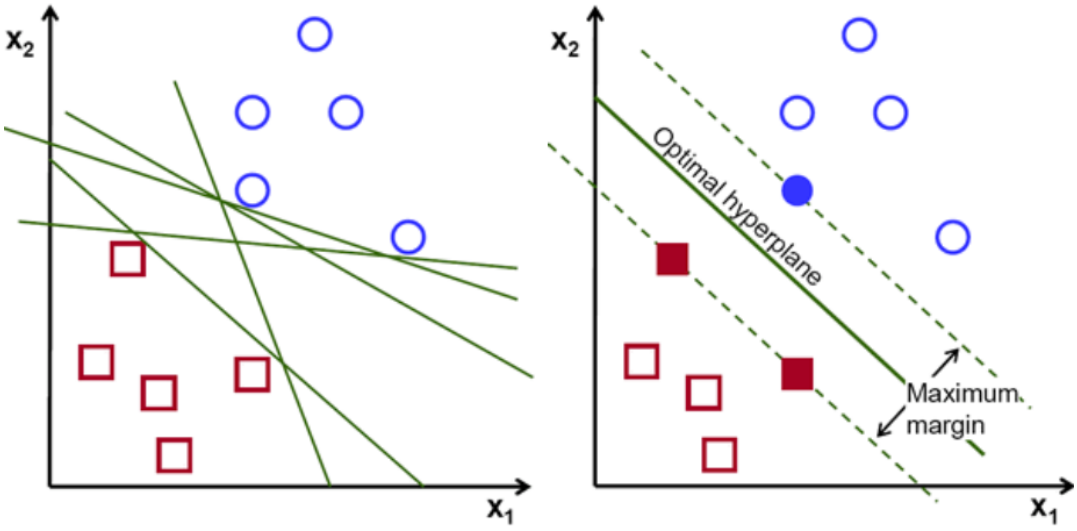
În partea **stângă** sunt reprezentate datele 2D originale. În **mijloc,** sunt reprezentate datele normalizate folosind norma L_2. În partea **dreaptă ,** sunt reprezentate datele normalizate folosind norma L_1.

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

Folosind norma L_1: $x\_scaled = \frac{X}{||X||_1}, ||X||_1 = \sum_{i=1}^{n} |x_i|$

Folosind norma L_2: $x\_scaled = \frac{X}{||X||_2}, ||X||_2 = \sqrt{\sum_{i=1}^{n} x_i{}^2}$

3. Mașini cu vectori suport



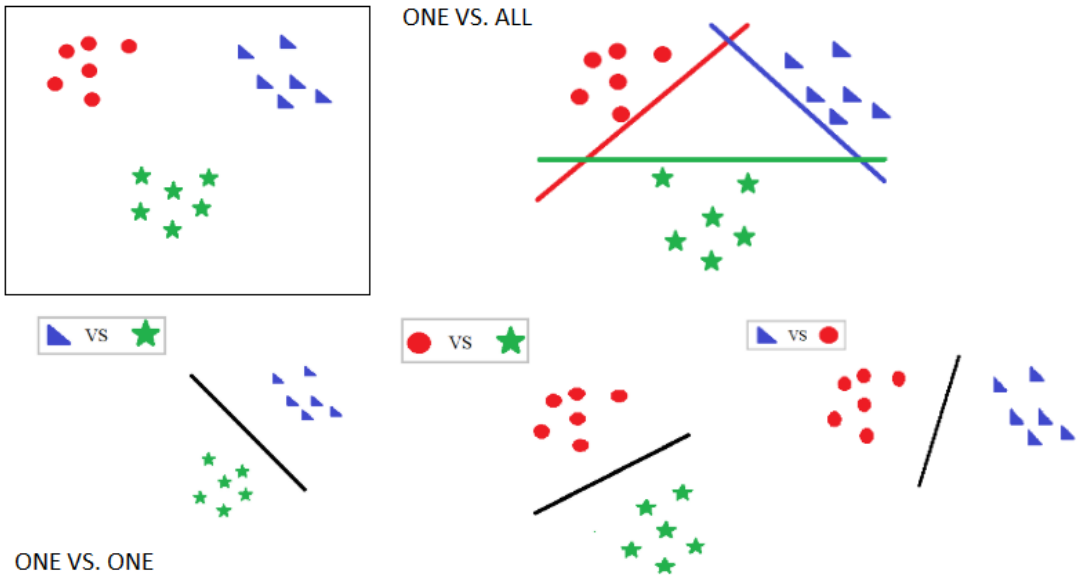
În partea stângă sunt prezentate drepte de decizie posibile pentru clasificarea celor două tipuri de obiecte. SVM-ul, exemplificat în partea dreaptă, alege hiperplanul care maximizează marginea dintre cele doua clase.

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.

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.

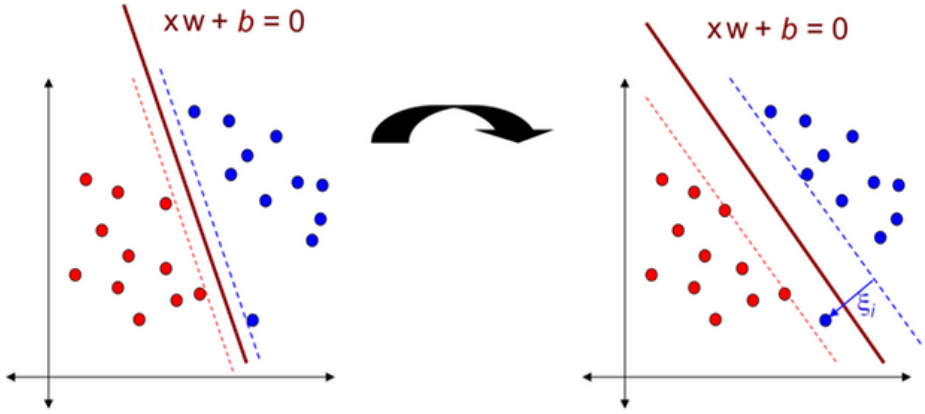
➔ La testare, clasa asignată fiecărui exemplu este cea care obține cele mai multe voturi pe baza acestor clasificatori.

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 supra-învățare (overfitting).

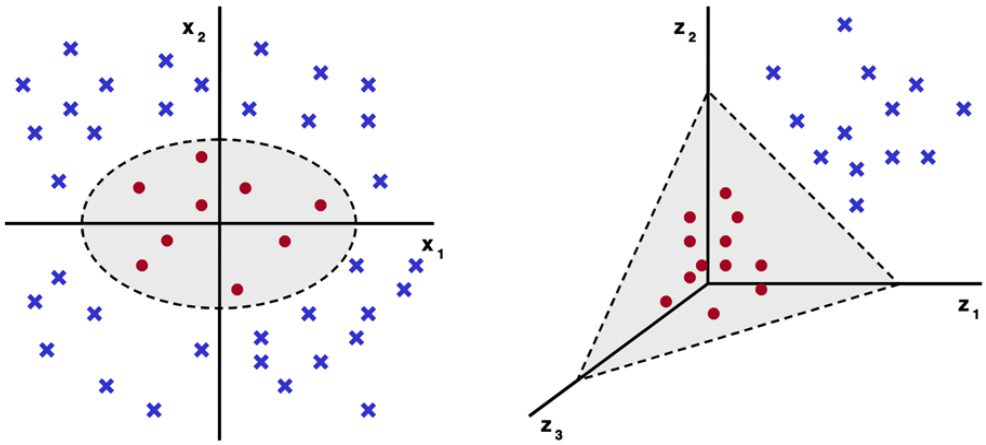
  - C mic - va fi ales un hiperplan cu o margine mai mare, chiar dacă acesta duce la clasificarea greșită a unor puncte din setul de antrenare (mai puțini vectori suport).

    Dacă C va fi ales prea mic, modelul nu va fi capabil să învețe, ajungându-se la sub-învățare (underfitting).

kernel (string, default = ‘linear’)

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$



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’

Sms Spam Classification

Această bază de date conține mesaje (sms) text spam/non-spam. Scopul nostru este să clasificăm dacă un mesaj este spam sau nu. Baza de date conține 3734 exemple de antrenare și 1840 exemple de testare. Raportul mesajelor non-spam:spam este de 6:1.

Exemple din setul de date:

spam URGENT! We are trying to contact you. Last weekends draw shows that you have won a £900 prize GUARANTEED. Call 09061701939. Claim code S89. Valid 12hrs only

ham Hi frnd, which is best way to avoid missunderstding wit our beloved one's?

ham Great escape. I fancy the bridge but needs her lager. See you tomo

Descărcați arhiva care conține datele de antrenare și testare de aici.

Exerciții

1. Descărcați arhiva ***de aici*** și observați cum funcționează modelul SVM.

2. 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.

3. Definiți clasa **BagOfWords** în al cărui constructor se inițializează vocabularul (un dicționar gol). În cadrul ei implementați metoda **build_vocabulary(self, data)** care primește ca parametru o listă de mesaje(listă de liste de strings) și construiește vocabularul pe baza acesteia. Cheile dicționarului sunt reprezentate de cuvintele din eseuri, iar valorile de id-urile unice atribuite cuvintelor. Pe lângă vocabularul pe care-l construiți, rețineți și o listă cu cuvintele în ordinea adăugării în vocabular.
   Afișați dimensiunea vocabularul construit (9522).

   OBS. Vocabularul va fi construit doar pe baza datelor din setul de antrenare.

4. Definiți metoda **get_features(self, data)** care primește ca parametru o listă de mesaje de dimensiune $num\_samples$(listă de liste de strings) și returnează o matrice de dimensiune ($num\_samples$ x $dictionary\_length$) definită astfel:

$$\boldsymbol{features(sample\_idx, word\_idx) = numarul\ de\ aparitii\ al\ cuvantului\ cu\ id - ul\ word\_idx\ in\ documentul\ sample\_idx}$$

5. Cu ajutorul funcțiilor definite anterior, obțineți reprezentările BOW pentru mulțimea de antrenare și testare, apoi normalizați-le folosind norma “L2”.

6. Antrenați un SVM cu ***kernel linear*** care să clasifice mesaje în mesaje spam/non-spam. Pentru parametrul **C** setați valoarea **1**. Calculați acuratețea și F1-score pentru mulțimea de testare.
   Afișați cele mai negative (spam) 10 cuvinte și cele mai pozitive (non-spam) 10 cuvinte.

   the first 10 negative words are ['Text' 'To' 'mobile' 'CALL' 'FREE' 'txt' '&' 'Call' 'Txt' 'STOP']
   the first 10 positive words are ['&lt#&gt' 'me' 'i' 'Going' 'him' 'Ok' 'I' 'Ill' 'my' 'Im']