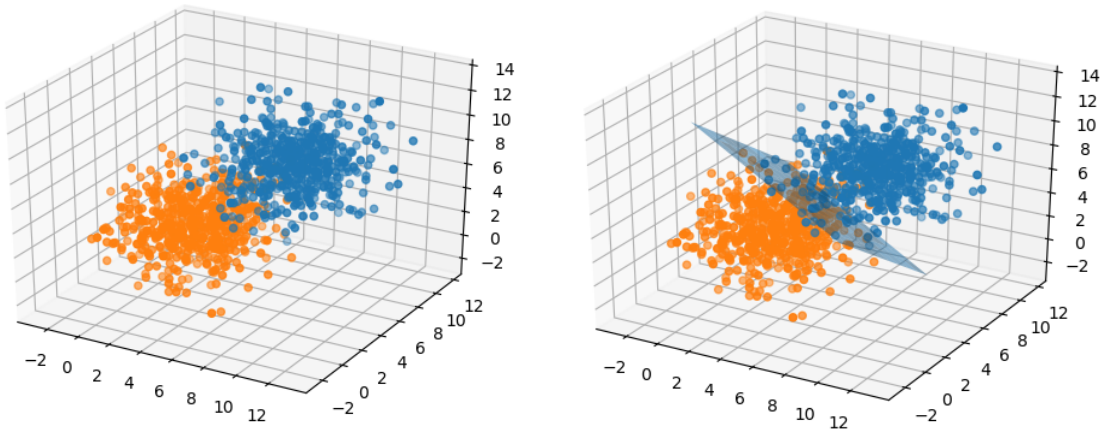


Perceptronul și rețele de perceptroni în Scikit-learn



Stanga:multimea de antrenare a punctelor 3d; Dreapta: multimea de testare a punctelor 3d si planul de separare.

În acest laborator vom antrena un perceptron cu ajutorul bibliotecii **Scikit-learn** pentru clasificarea unor date 3d, și o rețea neuronală pentru clasificarea cifrelor scrise de mână. Baza de date pe care o vom folosi, pentru clasificare cifrelor scrise de mână, este **MNIST**.

Multimile de antrenare și testare se găsesc [aici](#). Setul de date 3d, conține 1,000 de puncte 3d pentru antrenare, împărțite în 2 clase (1- pozitiv, -1 negativ) și 400 de puncte 3d pentru testare.

1. Definirea unui perceptron în Scikit-learn.

```
from sklearn.linear_model import Perceptron # importul clasei
perceptron_model = Perceptron(penalty=None, alpha=0.0001, fit_intercept=True,
max_iter=None, tol=None, shuffle=True, eta0=1.0, early_stopping=False,
validation_fraction=0.1, n_iter_no_change=5)
# toti parametrii sunt optionalii avand valori setate implicit
```

Parametri:

- **penalty (None, 'l2' sau 'l1' sau 'elasticnet', default=None):** metoda de regularizare folosita
- **alpha (float, default=0.0001):** parametru de regularizare.
- **fit_intercept (bool, default=True):** daca vrem sa invatam si bias-ului.
- **max_iter (int, default=5):** numarul maxim de epoci pentru antrenare.
- **tol (float, default=1e-3):**

- Dacă eroarea sau scorul nu se îmbunătățesc timp *n_iter_no_change* epoci consecutive cu cel puțin *tol*, antrenarea se oprește.
- **shuffle (bool, default=True)**: amesteca datele la fiecare epoca.
- **eta0 (double, default=1)**: rata de învățare.
- **early_stopping (bool, default=False)**:
 - Dacă este setat cu *True* atunci antrenarea se va termina dacă eroarea pe mulțimea de validare (care va fi setată automat) nu se îmbunătățește timp *n_iter_no_change* epoci consecutive cu cel puțin *tol*.
- **validation_fraction : (float, optional, default=0.1)**
 - Procentul din mulțimea de antrenare care va fi folosit pentru validare (doar când *early_stopping=True*). Trebuie să fie între 0 și 1.
- **n_iter_no_change (int, optional, default=5, sklearn-versiune-0.20)**:
 - Numărul maxim de epoci fără îmbunătățiri (eroare sau scor).

Funcțiile și atributele modelului:

- **perceptron_model.fit(X, y)**: antrenează clasificatorul utilizând stochastic gradient descent (algoritmul de coborare pe gradient), folosind parametri setați la definirea modelului
 - X - datele de antrenare, y - etichetele
 - X are dimensiunea (num_samples, num_features)
 - y are dimensiunea (num_features,)
 - returnează modelul antrenat.
- **perceptron_model.score(X, y)**: returnează acuratețea clasificatorului pe mulțimea de testare și etichetele primite ca argumente
- **perceptron_model.predict(X)**: returnează etichetele prezise de model
- **perceptron_model.coef_** : ponderile învățate
- **perceptron_model.intercept_** : bias-ul
- **perceptron_model.n_iter_**: numărul de epoci parcurse până la convergență

2. Definirea unei rețele de perceptroni în Scikit-learn.

```
from sklearn.neural_network import MLPClassifier # importul clasei

mlp_classifier_model = MLPClassifier(hidden_layer_sizes=(100, ),
activation='relu', solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=200, shuffle=True, random_state=None, tol=0.0001,
momentum=0.9, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=10)
```

Parametrii:

- `hidden_layer_sizes` (**tuple**, **lungime= $n_layers - 2$** , **default=(100,)**): al i -lea element reprezinta numarul de neuroni din al i -lea strat ascuns.
- `activation` ({**'identity'**, **'logistic'**, **'tanh'**, **'relu'**}, **default='relu'**)
 - **'Identity'**: $f(x) = x$
 - **'logistic'**: $f(x) = \frac{1}{1 + e^{-x}}$
 - **'tanh'**: $f(x) = \tanh(x)$
 - **'relu'**: $f(x) = \max(0, x)$
- `solver` ({**'lbfgs'**, **'sgd'**, **'adam'**}, **default='adam'**): regula de invatare (update)
 - **'sgd'** - stochastic gradient descent (doar pe acesta il vom folosi).
- `batch_size`: (**int**, **default='auto'**)
 - **auto** - marimea batch-ului pentru antrenare este $\min(200, n_samples)$.
- `learning_rate_init` (**double**, **default=0.001**): rata de invatare
- `max_iter` (**int**, **default=200**): numarul maxim de epoci pentru antrenare.
- `shuffle` (**bool**, **default=True**): amesteca datele la fiecare epoca
- `tol` (**float**, **default=1e-4**) :
 - Daca eroarea sau scorul nu se imbunatatesc timp $n_iter_no_change$ epoci consecutive (si $learning_rate \neq 'adaptive'$) cu cel putin tol , antrenarea se opreste.
- `n_iter_no_change` : (**int**, **optional**, **default 10**, *sklearn-versiune-0.20*)
 - Numarul maxim de epoci fara imbunatatiri (eroare sau scor).
- `alpha` (**float**, **default=0.0001**): parametru pentru regularizare L2.
- `learning_rate` ({**'constant'**, **'invscaling'**, **'adaptive'**}, **default='constant'**):
 - **'constant'** : rata de invatare este constanta si este data de parametrul $learning_rate_init$.
 - **'invscaling'**: rata de invatare va fi scazuta la fiecare pas t , dupa formula: $new_learning_rate = learning_rate_init / \text{pow}(t, power_t)$
 - **'adaptive'**: pastreaza rata de invatare constanta cat timp eroarea scade. Daca eroarea nu scade cu cel putin tol (fata de epoca anterior) sau daca scorul pe multimea de validare (*doar daca $early_stopping=True$*) nu creste cu cel putin tol (fata de epoca anterior), rata de invatare curenta se imparte la 5.
- `power_t` (**double**, **default=0.5**): parametrul pentru $learning_rate='invscaling'$.
- `momentum` (**float**, **default=0.9**): - valoarea pentru momentum cand se foloseste gradient descent cu momentum. Trebuie sa fie intre 0 si 1.
- `early_stopping` (**bool**, **default=False**):
 - Daca este setat cu *True* atunci antrenarea se va termina daca eroarea pe multimea de validare nu se imbunatateste timp $n_iter_no_change$ epoci consecutive cu cel putin tol .
- `validation_fraction` (**float**, **optional**, **default=0.1**):
 - Procentul din multimea de antrenare care sa fie folosit pentru validare (doar cand $early_stopping=True$). Trebuie sa fie intre 0 si 1.

Atribute:

- **classes_** : array sau o lista de array de dimensiune (n_classes,)
 - Clasele pentru care a fost antrenat clasificatorul.
- **loss_** : float, eroarea actuala
- **coefs_** : lista, lungimea = n_layers - 1
 - Al *i*-lea element din lista reprezinta matricea de ponderi dintre stratul *i* si *i + 1*.
- **intercepts_** : lista, lungimea n_layers - 1
 - Al *i*-lea element din lista reprezinta vectorul de bias corespunzator stratului *i + 1*.
- **n_iter_** : int, numarul de epoci parcurse pana la convergenta.
- **n_layers_** : int, numarul de straturi.
- **n_outputs_** : int, numarul de neuroni de pe stratul de iesire.
- **out_activation_** : string, numele functiei de activare de pe stratul de iesire.

Functii:

- **mlp_classifier_model.fit(X, y):**
 - Antreneaza modelul pe datele de antrenare X si etichetele y cu parametrii setati la declarare.
 - X este o matrice de dimensiune (n_samples, n_features).
 - y este un vector sau o matrice de dimensiune (n_samples,) - pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclass.
 - Returneaza modelul antrenat.
- **mlp_classifier_model.predict(X):**
 - Prezice etichetele pentru X folosind ponderile invatate.
 - X este o matrice de dimensiune (n_samples, n_features).
 - Returneaza clasele prezise intr-o matrice de dimensiune (n_samples,)- pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclass.
- **mlp_classifier_model.predict_proba(X):**
 - Prezice probabilitatea pentru fiecare clasa.
 - X este o matrice de dimensiune (n_samples, n_features).
 - Returneaza o matrice de (n_samples, n_classes) avand pentru fiecare exemplu si pentru fiecare clasa probabilitatea ca exemplul sa se afle in clasa respectiva.
- **mlp_classifier_model.score(X, y):**
 - Returneaza acurateta medie in functie de X si y.
 - X este o matrice de dimensiune (n_samples, n_features).
 - y are dimensiunea (n_samples,) - pentru clasificare binara si regresie, (n_samples, n_outputs) pentru clasificare multiclass.

Exerciții

1. Antrenati un perceptron pe multimea de puncte 3d, pana cand eroare nu se imbunatateste cu $1e-5$ fata de epocile anterioare, cu rata de invatare 0.1. Calculati acuratetea pe multimea de antrenare si testare, apoi afisati ponderile, bias-ul si numarul de epoci parcurse pana la convergenta. Plotati planul de decizie al clasicatorului cu ajutorului functiei `plot3d_data_and_decision_function`.

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def plot3d_data_and_decision_function(X, y, W, b):
    ax = plt.axes(projection='3d')
    # create x,y
    xx, yy = np.meshgrid(range(10), range(10))
    # calculate corresponding z
    # [x, y, z] * [W[0], W[1], W[2]] + b = 0
    zz = (-W[0] * xx - W[1] * yy - b) / W[2]
    ax.plot_surface(xx, yy, zz, alpha=0.5)
    ax.scatter3D(X[y == -1, 0], X[y == -1, 1], X[y == -1, 2], 'b');
    ax.scatter3D(X[y == 1, 0], X[y == 1, 1], X[y == 1, 2], 'r');
    plt.show()
```

2. Antrenati o retea de perceptroni care sa clasifice cifrele scrise de mana MNIST. Datele trebuie normalizate prin scaderea mediei si impartirea la deviatia standard. Antrenati si testati urmatoarele configuratii de retele:

- a. Functia de activare 'tanh', hidden_layer_sizes=(1), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iter=200 (default)
- b. Functia de activare 'tanh', hidden_layer_sizes=(10), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iter=200 (default)
- c. Functia de activare 'tanh', hidden_layer_sizes=(10), learning_rate_init=0.00001, momentum=0 (nu vom folosi momentum), max_iter=200 (default)
- d. Functia de activare 'tanh', hidden_layer_sizes=(10), learning_rate_init=10, momentum=0 (nu vom folosi momentum), max_iter=200 (default)
- e. Functia de activare 'tanh', hidden_layer_sizes=(10), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iters=20

- f. Functia de activare 'tanh', hidden_layer_sizes=(10, 10), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iter=2000
- g. Functia de activare 'relu', hidden_layer_sizes=(10, 10), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iter=2000
- h. Functia de activare 'relu', hidden_layer_sizes=(100, 100), learning_rate_init=0.01, momentum=0 (nu vom folosi momentum), max_iter=2000
- i. Functia de activare 'relu', hidden_layer_sizes=(100, 100), learning_rate_init=0.01, momentum=0.9, max_iter=2000
- j. Functia de activare 'relu', hidden_layer_sizes=(100, 100), learning_rate_init=0.01, momentum=0.9, max_iter=2000, alpha=0.005)