

# Inteligența artificială (cu Python)

- pentru ingineri roboticieni -

Stelian Brad



Învățare interactivă

# Lucrarea 1: Testarea perceptronului

# Generarea de date pentru efectuarea exercitiilor

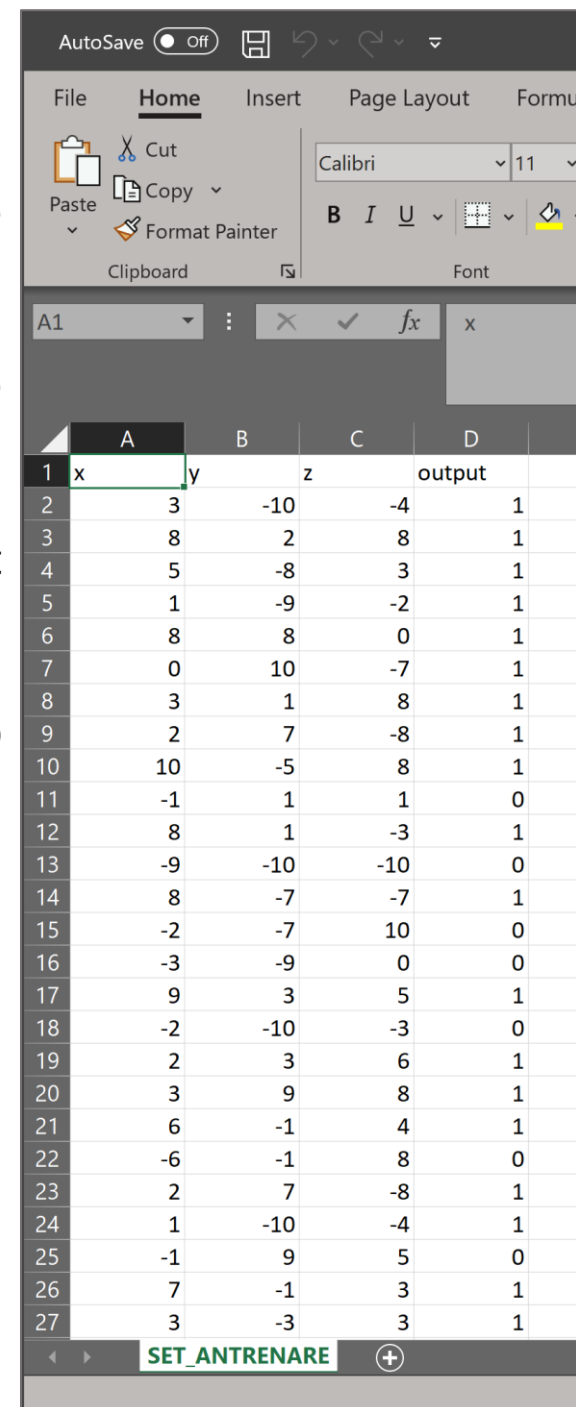
Folositi Excelul ca sa creati fisiere csv. Incepeti prin a va crea capul de tabel. In exemplul de aici am creat x, y, z, output.

Intrati in prima caseta A2 si =RANDBETWEEN(val1,val2). Aici am creat valori aleatorii intre val1=-10 si val2=10, de tip intreg. Faceti lucrul respectiv pentru toate coloanele de intrare.

Pentru coloana de iesire creati conditia in functie de reguli ale intrarilor. Aici am creat conditia ca iesirea sa fie 0 pentru orice set de intrare in care  $x < 0$  (=IF(A2<0, 0,1)).

Pentru a va exersa puteti sa creati orice date de intrare si iesire, reale, intregi sau de tip string. Consultati MS Excel pentru detalii pe acest subiect.

In practica operam cu date reale.



	A	B	C	D
1	x	y	z	output
2	3	-10	-4	1
3	8	2	8	1
4	5	-8	3	1
5	1	-9	-2	1
6	8	8	0	1
7	0	10	-7	1
8	3	1	8	1
9	2	7	-8	1
10	10	-5	8	1
11	-1	1	1	0
12	8	1	-3	1
13	-9	-10	-10	0
14	8	-7	-7	1
15	-2	-7	10	0
16	-3	-9	0	0
17	9	3	5	1
18	-2	-10	-3	0
19	2	3	6	1
20	3	9	8	1
21	6	-1	4	1
22	-6	-1	8	0
23	2	7	-8	1
24	1	-10	-4	1
25	-1	9	5	0
26	7	-1	3	1
27	3	-3	3	1

# Antrenarea perceptronului cu functia "rampa"

```
import pandas as pd
import numpy as np

input_dim = 3 #numarul de intrari
learning_rate = 0.01
#start ponderi in mod aleator (3 valori)
Weights = np.random.rand(input_dim)

Weights[0] = 0.5
Weights[1] = 0.5
Weights[2] = 0.5

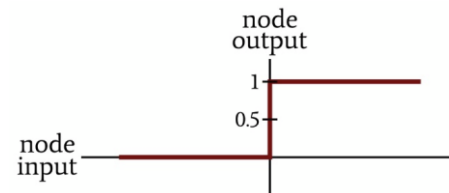
"""
Ponderile de la care se porneste influenteaza dramatic
ponderile finale. Aceasta este una dintre marile provocari
ale retelelor neuronale. De aceea, nu este recomandat pe
cat posibil sa pornesti cu valori alese aleatoriu"""

# extragem datele de instruire
Training_Data = pd.read_csv("C:\\BRAD_S\\BRAD\\SET_ANTRENARE.csv")
print(Training_Data)
Expected_Output = Training_Data.output
Training_Data = Training_Data.drop(['output'], axis=1)
Training_Data = np.asarray(Training_Data)

training_count = len(Training_Data[:,0])

for epoch in range(0,5):
    for datum in range(0, training_count):
        Output_Sum = np.sum(np.multiply(Training_Data[datum,:], Weights))
        # functia de activare de tip treapta
        if Output_Sum < 0:
            Output_Value = 0
        else:
            Output_Value = 1
        error = Expected_Output[datum] - Output_Value
        for n in range(0, input_dim):
            Weights[n] = Weights[n] + learning_rate*error*Training_Data[datum,n]

print("w_0 = %.3f" %(Weights[0]))
print("w_1 = %.3f" %(Weights[1]))
print("w_2 = %.3f" %(Weights[2]))
```



```
import pandas as pd
import numpy as np

input_dim = 3 #numarul de intrari
learning_rate = 0.01
#start ponderi in mod aleator (3 valori)
Weights = np.random.rand(input_dim)

"""
Weights[0] = 0.5
Weights[1] = 0.5
Weights[2] = 0.5
"""
```

Squeezed text (64 lines).

```
w_0 = 0.740
w_1 = 0.020
w_2 = 0.020
>>> |
```

Squeezed text (64 lines).

```
w_0 = 1.248
w_1 = 0.065
w_2 = 0.072
>>> |
```

= RESTART: C:\Users\steli\Ap

Squeezed text (64 lines).

```
w_0 = 0.677
w_1 = 0.003
w_2 = 0.034
>>> |
```

= RESTART: C:\Users\steli\Ap

Squeezed text (64 lines).

```
w_0 = 1.184
w_1 = 0.031
w_2 = 0.094
>>> |
```

= RESTART: C:\Users\steli\Ap

Squeezed text (64 lines).

```
w_0 = 0.804
w_1 = 0.012
w_2 = 0.002
>>> |
```

= RESTART: C:\Users\steli\Ap

Squeezed text (64 lines).

```
w_0 = 0.507
w_1 = 0.023
w_2 = 0.002
>>> |
```

Se observa ca ponderile finale ale nodurilor din retea sunt influentate de setul de valori de start.

Rata de învățare (learning rate) influențează rata la care învață rețeaua neuronală.

Se antrenează o rețea neuronală oferind date de instruire și efectuând o procedură de instruire. În timp ce acest lucru se întâmplă, rețeaua învață - sau mai precis, învață să aproximeze relația intrare-ieșire conținută în datele de instruire.

Manifestarea învățării este modificarea greutateii (ponderii), iar rata de învățare afectează modul în care greutățile sunt modificate.

# Antrenarea perceptronului cu functia "rampa"

```
import pandas as pd
import numpy as np

input_dim = 3 #numarul de intrari
learning_rate = 0.01
Weights = [0 for x in range(input_dim)]
Weights[0] = 0.5
Weights[1] = 0.5
Weights[2] = 0.5
# extragem datele de instruire
Training_Data = pd.read_csv("C:\\BRAD\\S\\BRAD\\SET_ANTRENARE.csv")
Expected_Output = Training_Data.output
Training_Data = Training_Data.drop(['output'], axis=1)
Training_Data = np.asarray(Training_Data)

training_count = len(Training_Data[:,0])
# numarul de date pe prima coloana
print("Numarul de elemente in lista de instruire este", training_count)
#punem numarul de epoci la 100
for epoch in range(0,5):
    for datum in range(0, training_count):
        Output_Sum = np.sum(np.multiply(Training_Data[datum,:], Weights))
        # functia de activare de tip treapta
        if Output_Sum < 0:
            Output_Value = 0
        else:
            Output_Value = 1
        error = Expected_Output[datum] - Output_Value
        for n in range(0, input_dim):
            Weights[n] = Weights[n] + learning_rate*error*Training_Data[datum,n]

print("w_0 = %.3f" %(Weights[0]))
print("w_1 = %.3f" %(Weights[1]))
print("w_2 = %.3f" %(Weights[2]))

def aplicare(d_intrare):
    element = 0
    for i in range(0,input_dim):
        element = element + d_intrare[i]*Weights[i]
    # functia de activare
    if element <0:
        iesire=0
    else:
        iesire=1
    print("Pentru intrarea", d_intrare, ", iesirea este", iesire)

intrare1 = [2, -5, 8]
intrare2 = [-3,4,-2]
aplicare(intrare1)
aplicare(intrare2)
```

Initializam o lista

Cresterea numarului de epoci nu va duce la rezultate mai bune. Totul depinde de acuratete.

Antrenam reteaua

Aici am folosit doar 5 iteratii (epoci) pentru antrenare

Testam reteaua

```
Numarul de elemente in lista de instruire este 66
w_0 = 0.740
w_1 = 0.020
w_2 = 0.020
Pentru intrarea [2, -5, 8] , iesirea este 1
Pentru intrarea [-3, 4, -2] , iesirea este 0
>>> |
```

```
Numarul de elemente in lista de instruire este 66
Numar epoci 100
w_0 = 1.370
w_1 = 0.010
w_2 = 0.150
Pentru intrarea [2, -5, 8] , iesirea este 1
Pentru intrarea [-3, 4, -2] , iesirea este 0
>>> |
```

```
Numarul de elemente in lista de instruire este 66
Numar epoci 1000
w_0 = 2.230
w_1 = 0.050
w_2 = 0.100
Pentru intrarea [2, -5, 8] , iesirea este 1
Pentru intrarea [-3, 4, -2] , iesirea este 0
>>> |
```

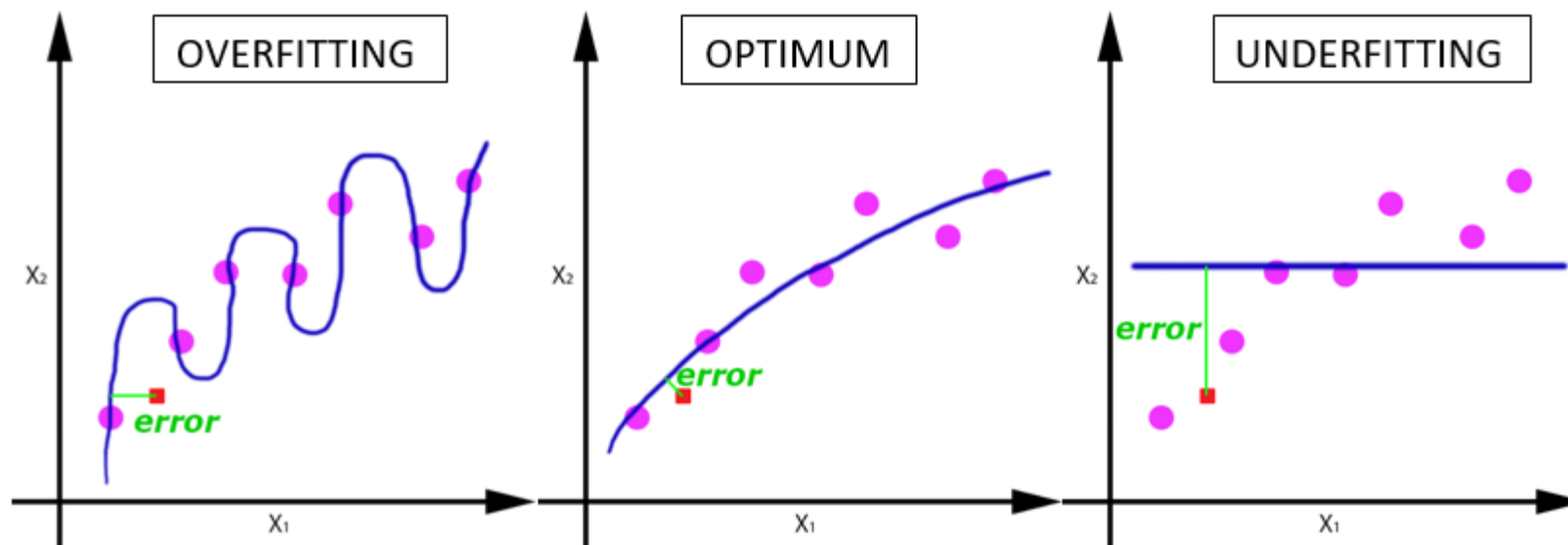
```
Numarul de elemente in lista de instruire este 66
Numar epoci 10000
w_0 = 2.230
w_1 = 0.050
w_2 = 0.140
Pentru intrarea [2, -5, 8] , iesirea este 1
Pentru intrarea [-3, 4, -2] , iesirea este 0
>>> |
```

```
Numarul de elemente in lista de instruire este 66
Numar epoci 100000
w_0 = 2.230
w_1 = 0.050
w_2 = 0.120
Pentru intrarea [2, -5, 8] , iesirea este 1
Pentru intrarea [-3, 4, -2] , iesirea este 0
>>> |
```

# Antrenarea perceptronului cu functia “rampa”

```
Numarul de elemente in lista de instruire este 66  
Numar epoci 100  
w_0 = 1.370  
w_1 = 0.010  
w_2 = 0.150  
Pentru intrarea [2, -5, 8] , iesirea este 1  
Pentru intrarea [-3, 4, -2] , iesirea este 0  
Pentru intrarea [-20, -20, 11] , iesirea este 0  
Pentru intrarea [-200, 2000, 1] , iesirea este 0  
Pentru intrarea [-200, 200000, 10000] , iesirea este 1  
>>>
```

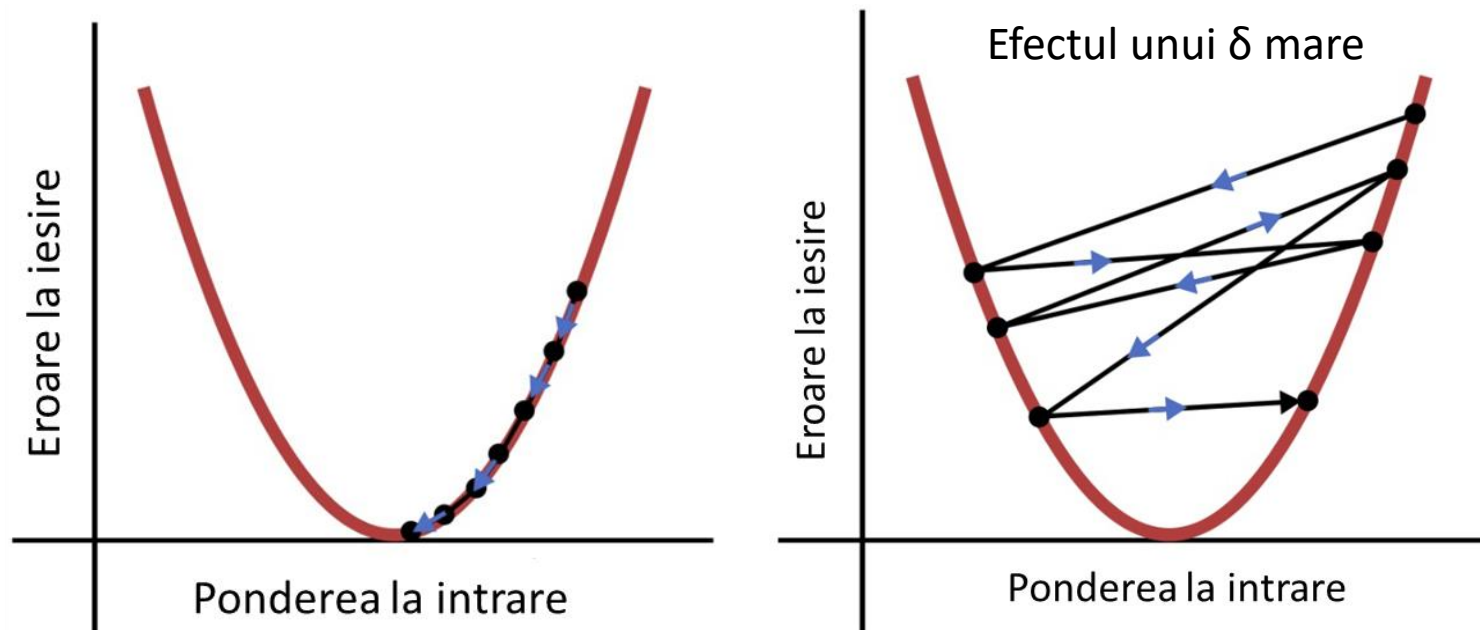
Observam ca atunci cand datele de intrare difera semnificativ de setul de antrenare, retea da un rezultat eronat



Numarul neadecvat de epoci poate conduce la diverse situatii, cum ar fi “overfitting”-ul, adica sa propui un model care nu este suficient de robust la un set mai divers de date de intrare.



# Antrenarea perceptronului cu functia “rampa”



Prin rata de invatare facem practic “salturi” in modificarea ponderilor. Scopul final este ca sa ajungem la eroarea minima. In functie de valoarea ratei de invatare “salturile” difera. Trebuie evitat cazul in care sarim peste minimul global. Nu exista o regula dupa care se alege rata de invatare! Nu exista inca nici o regula pentru a identifica rata de invatare optima pentru o aplicatie particulara! Totul se bazeaza pe intuitie si experienta. De aceea, setarea retelelor neuronale trebuie sa implice experti in domeniul de aplicare, nu programatori.

Regula de invatare:

$$w_{nou} = w + (\alpha \times \delta \times input)$$

- $\alpha$  este rata de învățare
- $\delta$  este diferența dintre rezultatul așteptat și rezultatul calculat (adică eroarea)

De fiecare dată când aplicăm regula de învățare, greutatea sare într-un punct nou pe curba de eroare. Dacă  $\delta$  este mare, aceste salturi ar putea fi, de asemenea, destul de mari și este posibil ca rețeaua să nu se antreneze eficient, deoarece greutățile nu converg treptat către erori minime. În schimb, salturile devin haotice.

# Exercitiu individual

Repetati aplicatia pentru alte functii de activare si trargeti concluzii. Folositi pe rand functiile de activare: sigmoid si ReLU.

**Optional:** studiatu codul atasat.

```
# Make a prediction with weights
def predict(row, weights):
    activation = weights[0]
    for i in range(len(row)-1):
        activation += weights[i + 1] * row[i]
    return 1.0 if activation >= 0.0 else 0.0

# Estimate Perceptron weights using stochastic gradient descent
def train_weights(train, l_rate, n_epoch):
    weights = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in train:
            prediction = predict(row, weights)
            error = row[-1] - prediction
            sum_error += error**2
            weights[0] = weights[0] + l_rate * error
            for i in range(len(row)-1):
                weights[i + 1] = weights[i + 1] + l_rate * error * row[i]
        print(>epoch=%d, lrate=0.3f, error=0.3f % (epoch, l_rate, sum_error))
    return weights
```

# perceptron\_src.py

```
# test predictions
dataset = [[2.7810836,2.550537003,0],
           [1.465489372,2.362125076,0],
           [3.396561688,4.400293529,0],
           [1.38807019,1.850220317,0],
           [3.06407232,3.005305973,0],
           [7.627531214,2.759262235,1],
           [5.332441248,2.088626775,1],
           [6.922596716,1.77106367,1],
           [8.675418651,-0.242068655,1],
           [7.673756466,3.508563011,1]]

weights = [-0.1, 0.20653640140000007, -0.23418117710000003]
for row in dataset:
    prediction = predict(row, weights)
    print("Expected=%d, Predicted=%d" % (row[-1], prediction))
```

```
# Calculate weights
dataset = [[2.7810836,2.550537003,0],
           [1.465489372,2.362125076,0],
           [3.396561688,4.400293529,0],
           [1.38807019,1.850220317,0],
           [3.06407232,3.005305973,0],
           [7.627531214,2.759262235,1],
           [5.332441248,2.088626775,1],
           [6.922596716,1.77106367,1],
           [8.675418651,-0.242068655,1],
           [7.673756466,3.508563011,1]]

l_rate = 0.1
n_epoch = 5
weights = train_weights(dataset, l_rate, n_epoch)
print(weights)
```

```
= RESTART: C:\Users\steli\AppData\Local\Programs\Py
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
>epoch=0, lrate=0.100, error=2.000
>epoch=1, lrate=0.100, error=1.000
>epoch=2, lrate=0.100, error=0.000
>epoch=3, lrate=0.100, error=0.000
>epoch=4, lrate=0.100, error=0.000
[-0.1, 0.20653640140000007, -0.23418117710000003]
>>>
```



# Studiu individual (optional)

Deschideti fisierele intr-un mediu compatibil cu scripturile Python.

perceptron\_sk.py

perceptron\_sonar.py

Pentru a doua aplicatie utilizati fisierul cu date: sonar.all-data.csv

Studiati codul si logica aplicatiei. Tragetii concluzii.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	0.02	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	0.1609	0.1582	0.2238	0.0645	0.066	0.2273	0.31	0.2999	0.5078	0.4797	0.5783
2	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	0.4918	0.6552	0.6919	0.7797	0.7464	0.9444	1	0.8874	0.8024	0.7818	0.5212
3	0.0262	0.0582	0.1099	0.1083	0.0974	0.228	0.2431	0.3771	0.5598	0.6194	0.6333	0.706	0.5544	0.532	0.6479	0.6931	0.6759	0.7551	0.8929	0.8619	0.7974
4	0.01	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	0.0881	0.1992	0.0184	0.2261	0.1729	0.2131	0.0693	0.2281	0.406	0.3973	0.2741
5	0.0762	0.0666	0.0481	0.0394	0.059	0.0649	0.1209	0.2467	0.3564	0.4459	0.4152	0.3952	0.4256	0.4135	0.4528	0.5326	0.7306	0.6193	0.2032	0.4636	0.4148
6	0.0286	0.0453	0.0277	0.0174	0.0384	0.099	0.1201	0.1833	0.2105	0.3039	0.2988	0.425	0.6343	0.8198	1	0.9988	0.9508	0.9025	0.7234	0.5122	0.2074
7	0.0317	0.0956	0.1321	0.1408	0.1674	0.171	0.0731	0.1401	0.2083	0.3513	0.1786	0.0658	0.0513	0.3752	0.5419	0.544	0.515	0.4262	0.2024	0.4233	0.7723
8	0.0519	0.0548	0.0842	0.0319	0.1158	0.0922	0.1027	0.0613	0.1465	0.2838	0.2802	0.3086	0.2657	0.3801	0.5626	0.4376	0.2617	0.1199	0.6676	0.9402	0.7832
9	0.0223	0.0375	0.0484	0.0475	0.0647	0.0591	0.0753	0.0098	0.0684	0.1487	0.1156	0.1654	0.3833	0.3598	0.1713	0.1136	0.0349	0.3796	0.7401	0.9925	0.9802
10	0.0164	0.0173	0.0347	0.007	0.0187	0.0671	0.1056	0.0697	0.0962	0.0251	0.0801	0.1056	0.1266	0.089	0.0198	0.1133	0.2826	0.3234	0.3238	0.4333	0.6068
11	0.0039	0.0063	0.0152	0.0336	0.031	0.0284	0.0396	0.0272	0.0323	0.0452	0.0492	0.0996	0.1424	0.1194	0.0628	0.0907	0.1177	0.1429	0.1223	0.1104	0.1847
12	0.0123	0.0309	0.0169	0.0313	0.0358	0.0102	0.0182	0.0579	0.1122	0.0835	0.0548	0.0847	0.2026	0.2557	0.187	0.2032	0.1463	0.2849	0.5824	0.7728	0.7852
13	0.0079	0.0086	0.0055	0.025	0.0344	0.0546	0.0528	0.0958	0.1009	0.124	0.1097	0.1215	0.1874	0.3383	0.3227	0.2723	0.3943	0.6432	0.7271	0.8673	0.9674
14	0.009	0.0062	0.0253	0.0489	0.1197	0.1589	0.1392	0.0987	0.0955	0.1895	0.1896	0.2547	0.4073	0.2988	0.2901	0.5326	0.4022	0.1571	0.3024	0.3907	0.3542
15	0.0124	0.0433	0.0604	0.0449	0.0597	0.0355	0.0531	0.0343	0.1052	0.212	0.164	0.1901	0.3026	0.2019	0.0592	0.239	0.3657	0.3809	0.5929	0.6299	0.5801
16	0.0298	0.0615	0.065	0.0921	0.1615	0.2294	0.2176	0.2033	0.1459	0.0852	0.2476	0.3645	0.2777	0.2826	0.3237	0.4335	0.5638	0.4555	0.4348	0.6433	0.3932
17	0.0352	0.0116	0.0191	0.0469	0.0737	0.1185	0.1683	0.1541	0.1466	0.2912	0.2328	0.2237	0.247	0.156	0.3491	0.3308	0.2299	0.2203	0.2493	0.4128	0.3158
18	0.0192	0.0607	0.0378	0.0774	0.1388	0.0809	0.0568	0.0219	0.1037	0.1186	0.1237	0.1601	0.352	0.4479	0.3769	0.5761	0.6426	0.679	0.7157	0.5466	0.5399
19	0.027	0.0092	0.0145	0.0278	0.0412	0.0757	0.1026	0.1138	0.0794	0.152	0.1675	0.137	0.1361	0.1345	0.2144	0.5354	0.683	0.56	0.3093	0.3226	0.443
20	0.0126	0.0149	0.0641	0.1732	0.2565	0.2559	0.2947	0.411	0.4983	0.592	0.5832	0.5419	0.5472	0.5314	0.4981	0.6985	0.8292	0.7839	0.8215	0.9363	1
21	0.0473	0.0509	0.0819	0.1252	0.1783	0.307	0.3008	0.2362	0.383	0.3759	0.3021	0.2909	0.2301	0.1411	0.1582	0.243	0.4474	0.5964	0.6744	0.7969	0.8319
22	0.0664	0.0575	0.0842	0.0372	0.0458	0.0771	0.0771	0.113	0.2353	0.1838	0.2869	0.4129	0.3647	0.1984	0.284	0.4039	0.5837	0.6792	0.6086	0.4858	0.3246
23	0.0099	0.0484	0.0299	0.0297	0.0652	0.1077	0.2363	0.2385	0.0075	0.1882	0.1456	0.1892	0.3176	0.134	0.2169	0.2458	0.2589	0.2786	0.2298	0.0656	0.1441
24	0.0115	0.015	0.0136	0.0076	0.0211	0.1058	0.1023	0.044	0.0931	0.0734	0.074	0.0622	0.1055	0.1183	0.1721	0.2584	0.3232	0.3817	0.4243	0.4217	0.4449
25	0.0293	0.0644	0.039	0.0173	0.0476	0.0816	0.0993	0.0315	0.0736	0.086	0.0414	0.0472	0.0835	0.0938	0.1466	0.0809	0.1179	0.2179	0.3326	0.3258	0.2111
26	0.0701	0.0076	0.0138	0.0062	0.0133	0.0151	0.0541	0.021	0.0505	0.1097	0.0841	0.0942	0.1204	0.042	0.0031	0.0162	0.0624	0.2127	0.3436	0.3813	0.3825

```
perceptron_sk.py - C:\Users\steli\AppData\Local\Programs\Python\Python37\pe...
File Edit Format Run Options Window Help
# evaluate a perceptron model on the dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=0, random_state=1)
# define model
model = Perceptron()
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

# make a prediction with a perceptron model on the dataset
from sklearn.datasets import make_classification
from sklearn.linear_model import Perceptron
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=10, n_redundant=0, random_state=1)
# define model
model = Perceptron()
# fit model
model.fit(X, y)
# define new data
row = [0.12777556, -3.64400522, -2.23268854, -1.82114386, 1.75466361, 0.1243966, 1.03397657, 2.351]
# make a prediction
yhat = model.predict([row])
# summarize prediction
print('Predicted Class: %d' % yhat)

# grid search learning rate for the perceptron
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron
# define dataset
```

# Stadiu individual: backpropagation – exemplificare cod Python

