Predicția geoefectivității ejecțiilor coronale de masă folosind algoritmi de învățare automată

Pricopi Andreea-Clara
Universitatea Tehnică din Cluj-Napoca
Dr. Diana Beșliu-Ionescu,
Dr. Alin Paraschiv (NSO)
Dr. Anca Mărginean (UTCN)

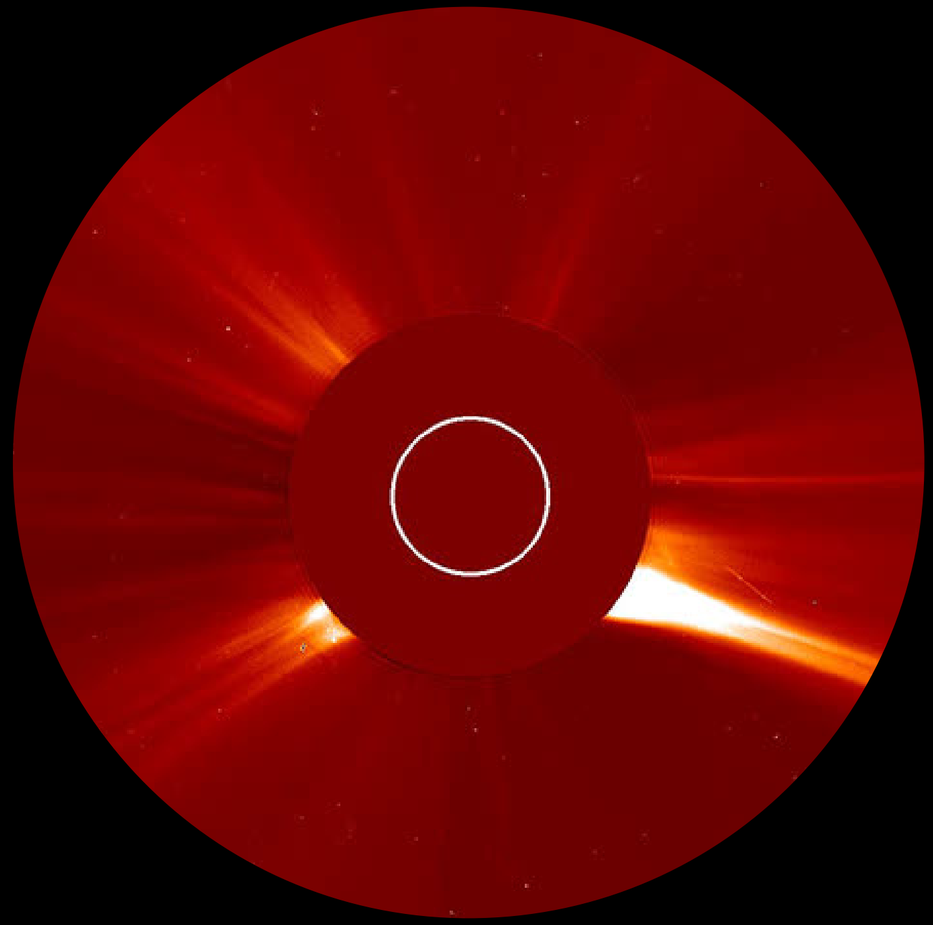
Subiectele de astăzi

Ce e o ejecție de masă coronală?

CME = Coronal Mass Ejection,

o explozie solară

care, în anumite circumstanțe, poate afecta Pământul

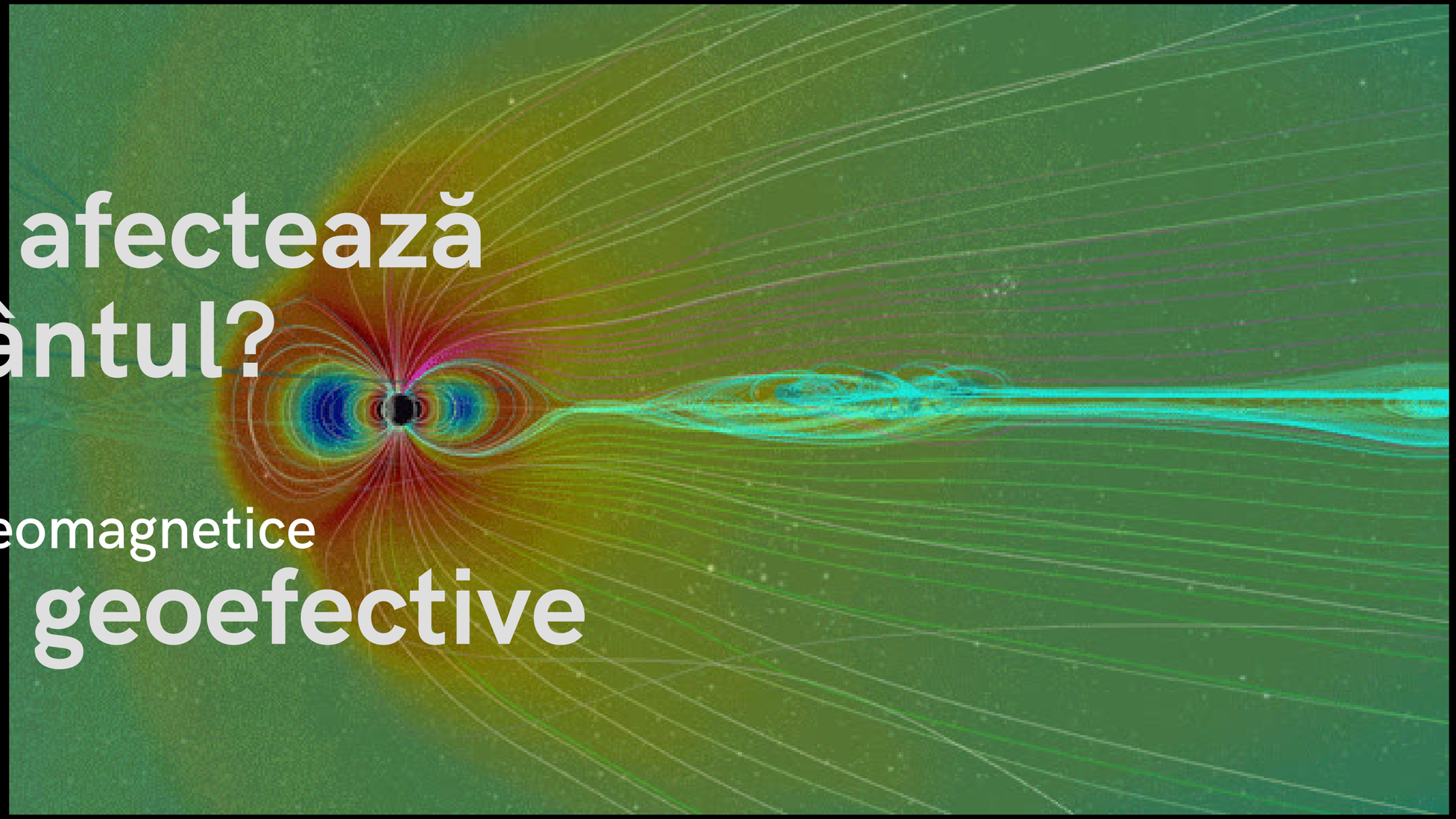


Cum afectează Pământul?

Generând

furtuni geomagnetice

i.e., fiind **geoefective**

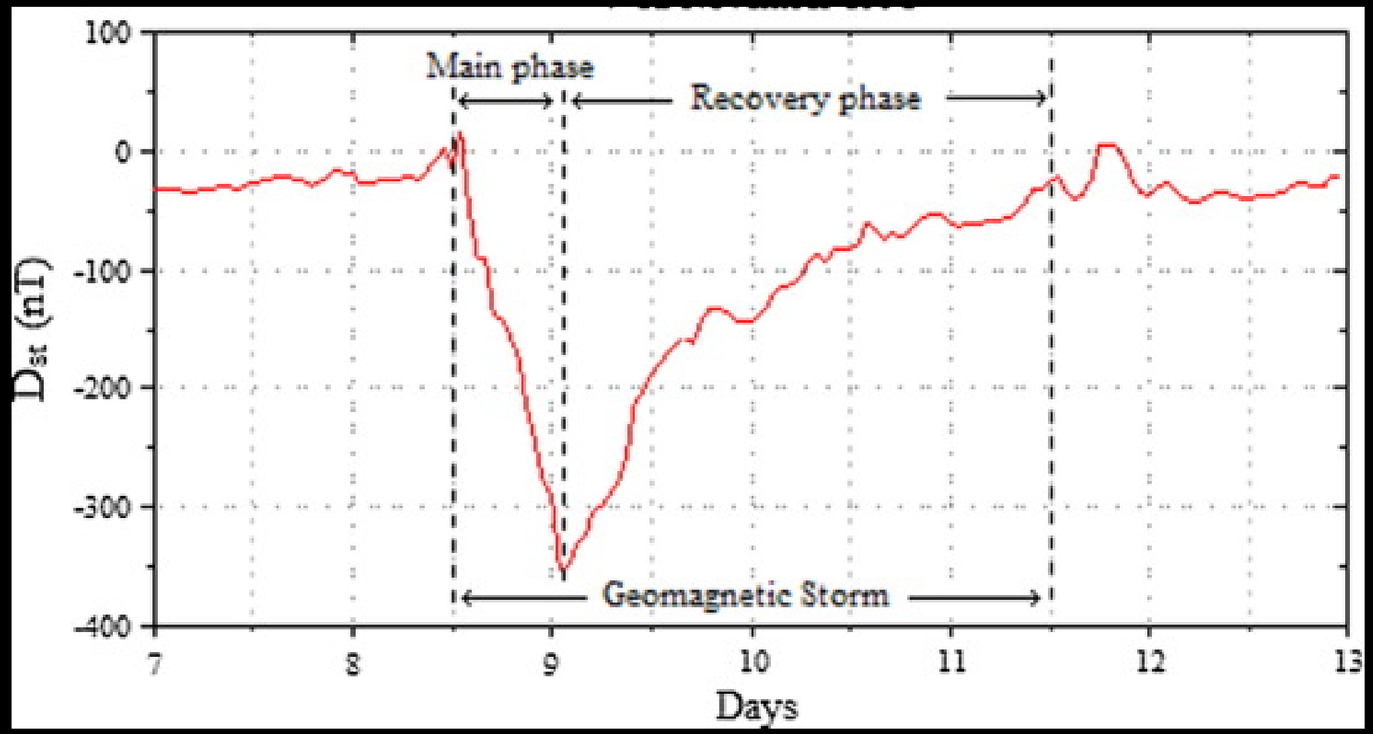


Cum știm dacă a avut loc o furtună geomagnetică?

Dst = Disturbance Storm Time,

(index geomagnetic)

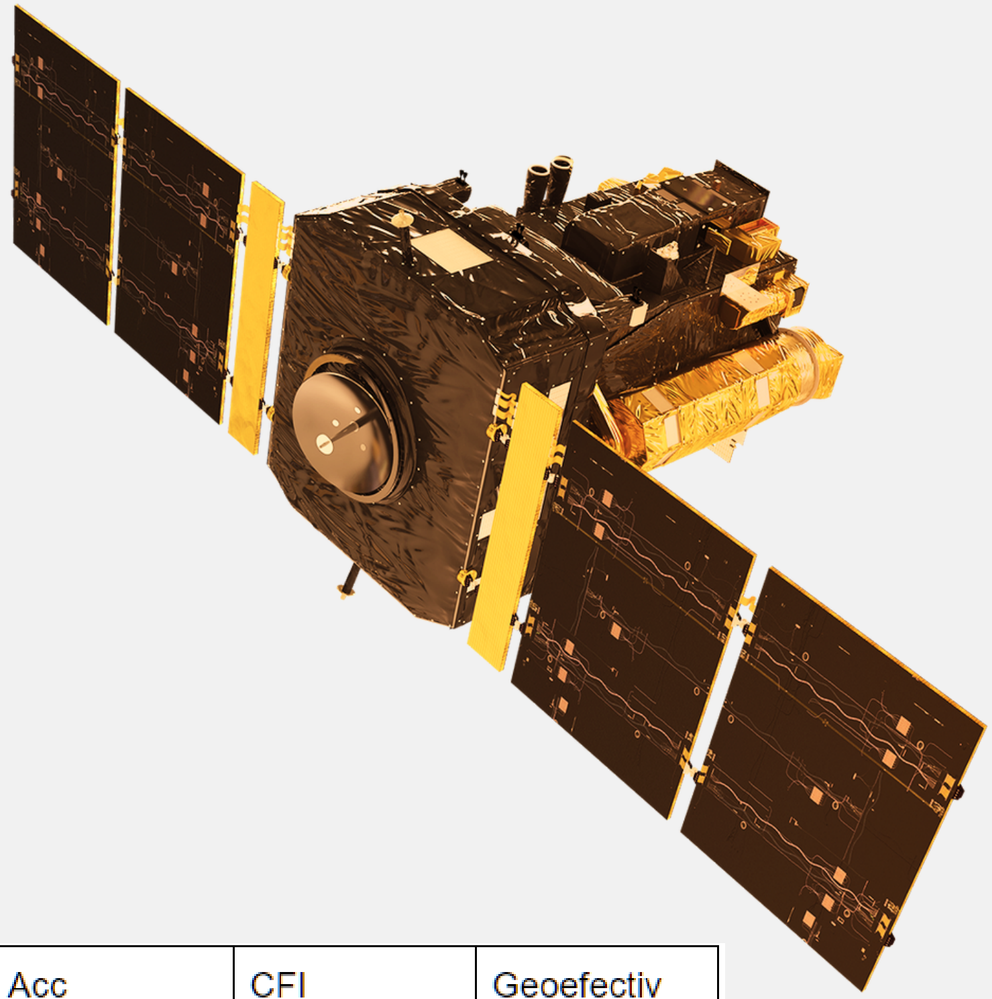
<= -30 nT



Ce își propune acest studiu?

Crearea, antrenarea și compararea a 5 modele de clasificare binară pentru a determina dacă un CME va fi asociat cu o valoare Dst <= -30 nT, utilizând doar parametri solari pentru a asigura timpii maximi de avertizare

01



Crearea setului de date

Date solare din catalogul SOHO LASCO
Corelarea cu valorile Dst din catalogul Richardson & Cane
Indexul cfi (Comprehensive Flare Index) NOAA

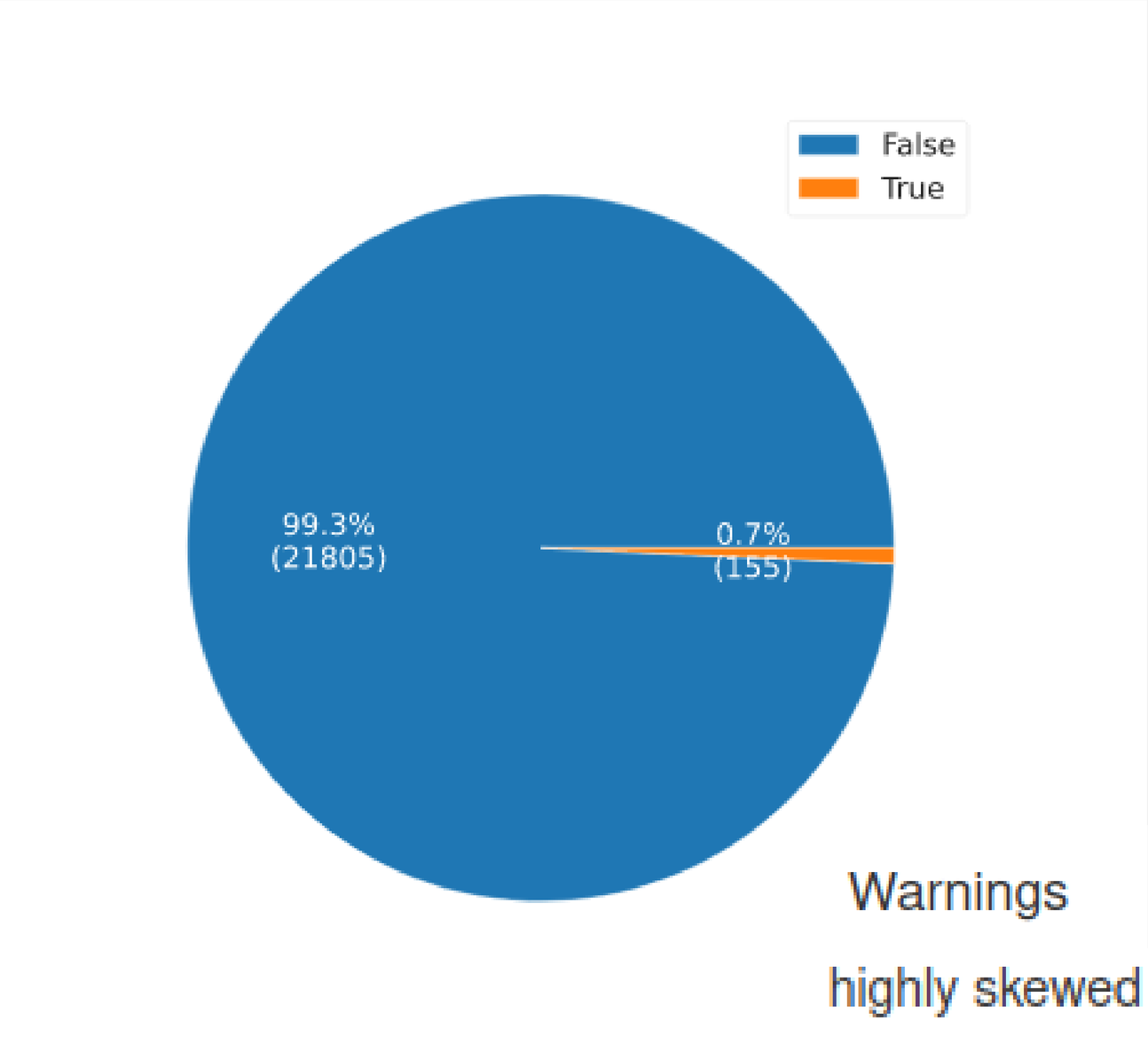
Convenție:
datele sunt tabelare
rânduri = „exemple”
coloane = „atribute”

atribut

exemplu	CPA	AW	Vlin	Acc	CFI	Geoefectiv
	360	360	1074	-26.9	54.3	0

02

Analiza setului de date și a problemelor



03

Înțelegerea problemelor

01

Dezechilibru extrem între clase

Peste 99% din evenimente nu au fost geoefective

02

Puțini parametri disponibili

Doar 5 variabile independente

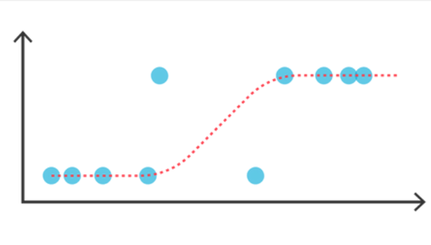
03

Ambiguitate?

ECM nu sunt singurele evenimente cu potențial geoefectiv
Uneori nu este sigur care ECM a fost cauza

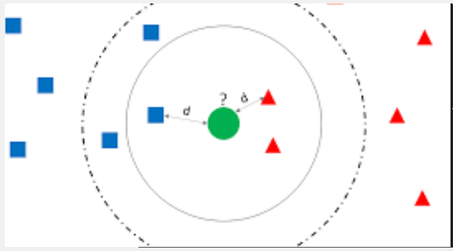
04

Algoritmi



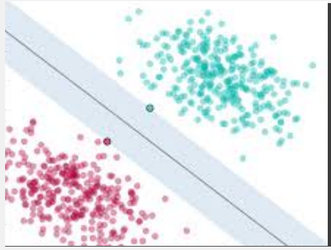
01

Regresie logistică



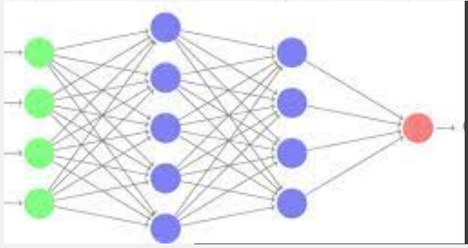
02

KNN



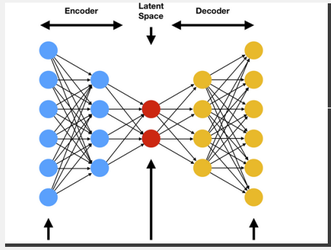
03

SVM



04

ANN



05

Autoencoders

Regresie logistică

Pentru varabilele de intrare X, vrem să obținem predicția ŷ

$$\hat{y} = P(y = 1|X)$$

ŷ este calculată după formulele din partea dreaptă
și ne dorim să fie cât mai aproape de valoarea reală y

$$\hat{y} = \sigma(W^T X + b)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Regresie logistică

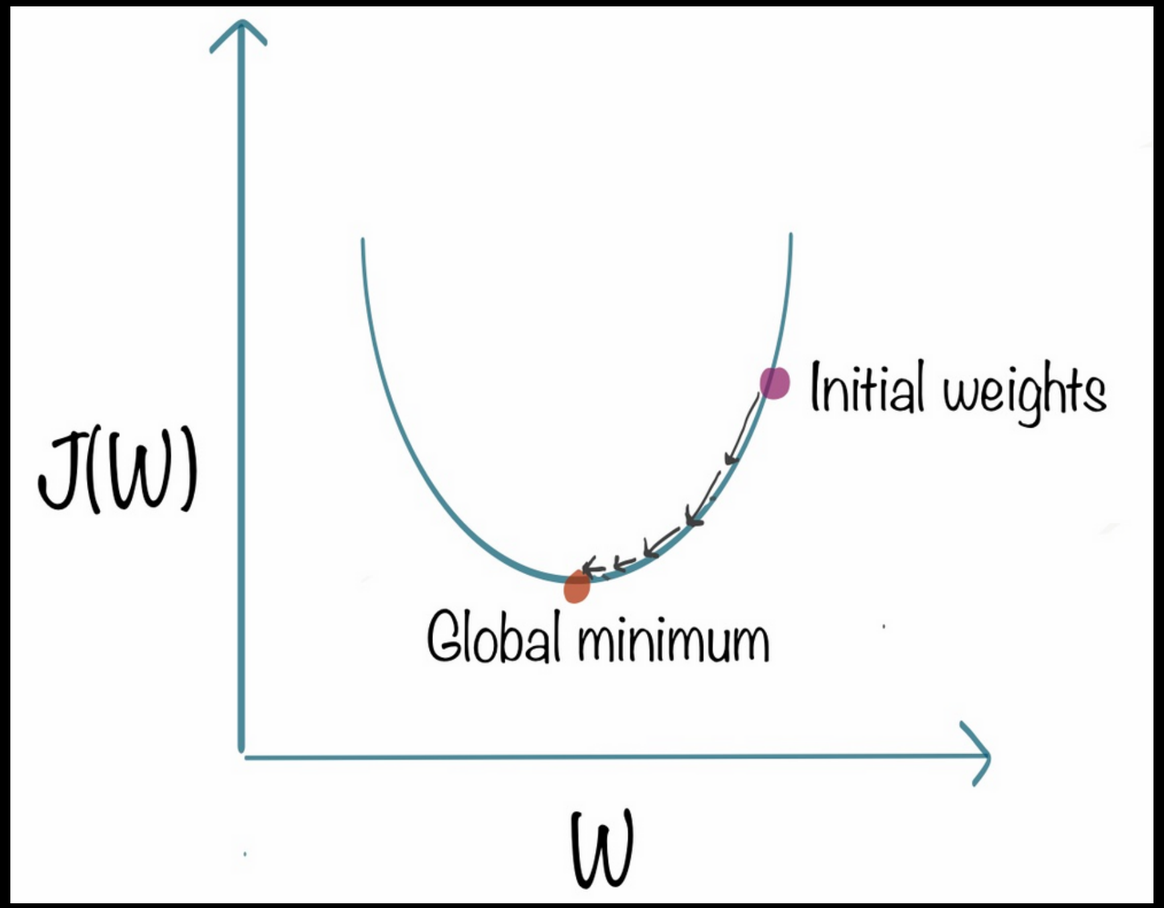
Cum definim "cât mai aproape de valoarea reală y" ?
Am avea nevoie de un mod de a calcula eroarea față de y, dorind să minimizăm această eroare

Regresie logistică

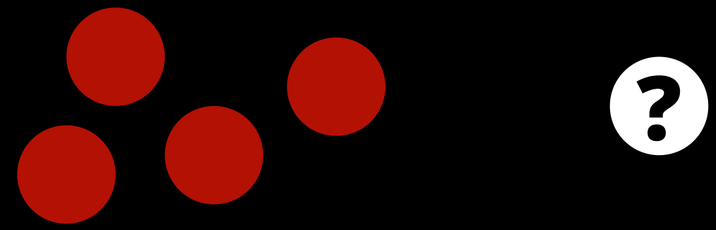
Modul de a calcula eroarea se numește funcție de cost J
Algoritmul modifică valorile W și b după fiecare calcul al erorii pentru a obține un cost minim

$$\mathcal{L}(\hat{y}, y) = -[y * log(\hat{y}) + (1 - y) * log(1 - \hat{y})]$$

$$J(W, b) = \frac{1}{m} * \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$



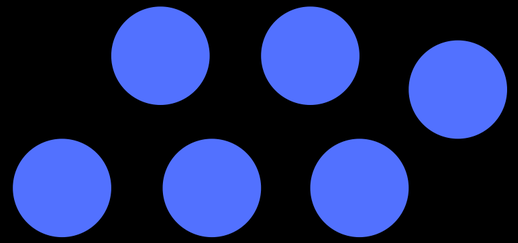
KNN (K-Nearest Neighbors)



"Cine se aseamănă se adună"

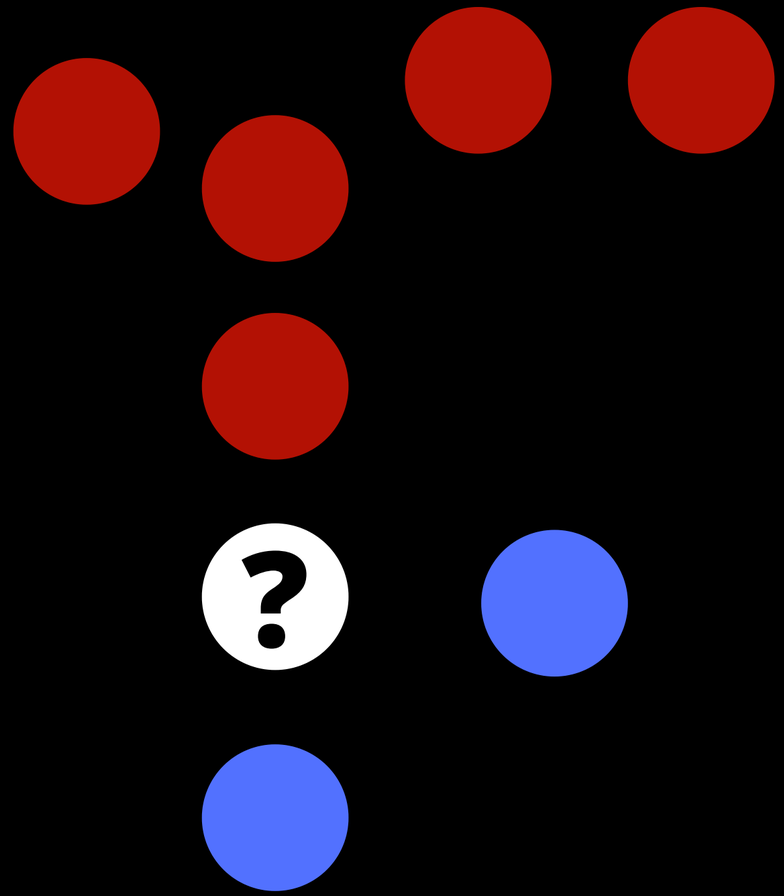
Distanța = măsură a similitudinii

Modul de calcul al distanței & K - specificate de noi
Eticheta = votul majoritar al celor K vecini



KNN (K-Nearest Neighbors)

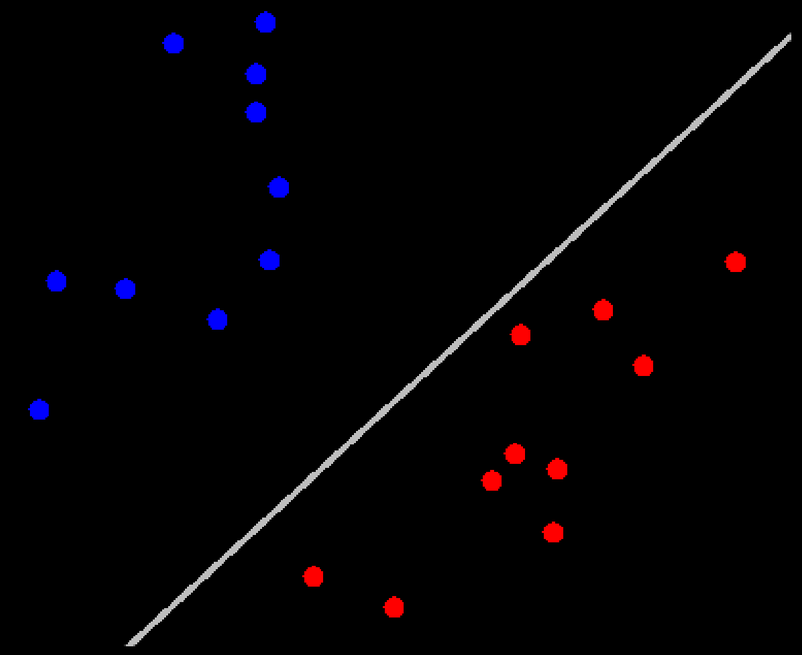
k = 1
--> egalitate, poate 1 singur vecin nu e relevant
k = 2
--> egalitate, nu avem majoritate
--> k ar trebui să fie impar
k = 3
--> albastru
k = 5
--> roșu



- putem experimenta cu diferiți k
- putem utiliza ponderi pentru distanțe (vecinii mai apropiați să conteze mai mult la vot)

SVM (Support Vector Machines)

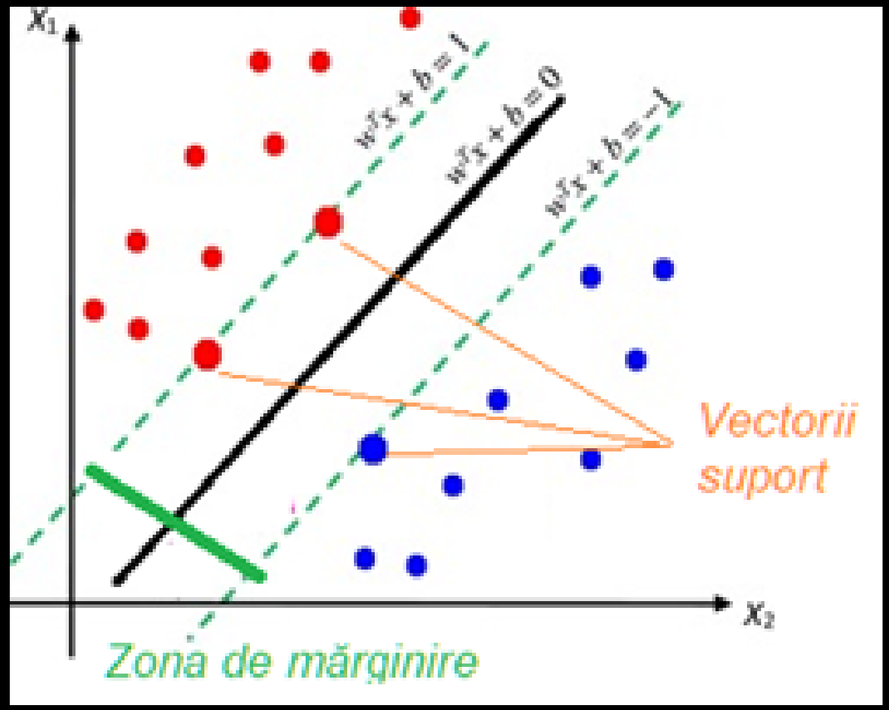
În câte moduri putem trage o linie care să despartă două clase liniar separabile?
Există un mod optim?



SVM (Support Vector Machines)

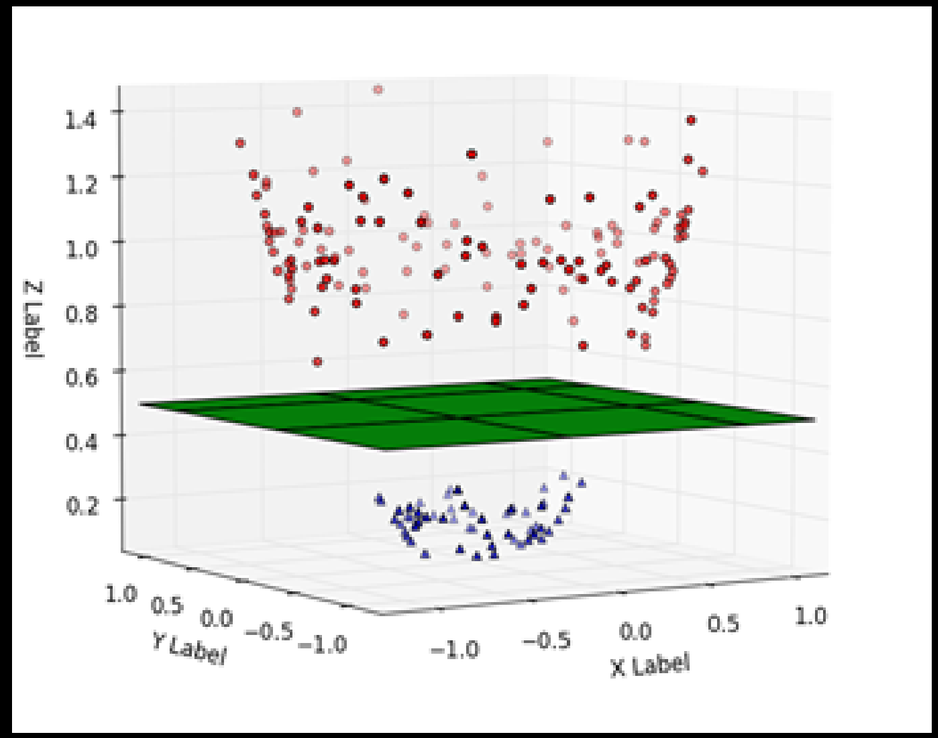
maximizarea zonei de mărginire
pentru că distanța până la dreapta de delimitare indică gradul de încredere

$$\hat{y} = \begin{cases} 0, W^T X + b < 0 \\ 1, W^T X + b \geq 0 \end{cases}$$



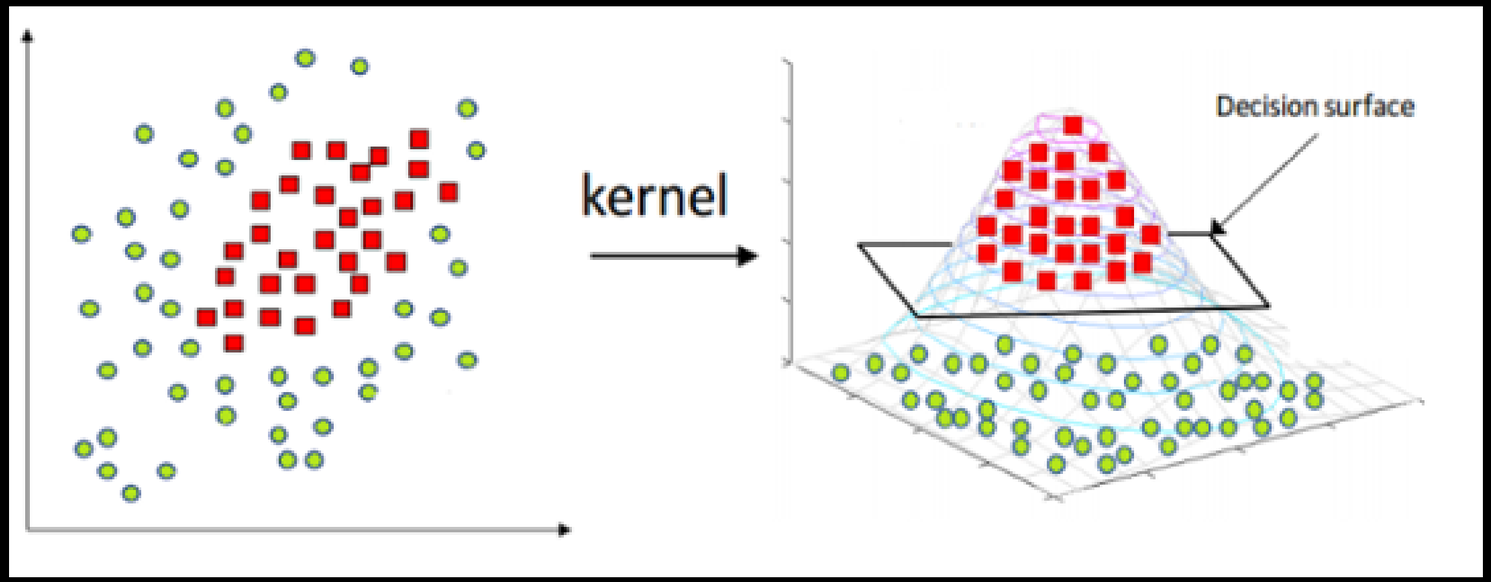
SVM (Support Vector Machines)

utilizat, de fapt, pentru
găsirea hiperplanului optim
pentru delimitarea a două clase



SVM (Support Vector Machines)

Dar nu întotdeauna datele sunt liniar separabile
Se consideră, însă, că există un spațiu n-dimensional în care ar fi



4| Agoritmi

SVM (Support Vector Machines)

Explicație oferită de Lili Jiang, Quora Data Science Manager

Originally Answered: What are Kernels in Machine Learning and SVM?

Briefly speaking, a kernel is a shortcut that helps us do certain calculation faster which otherwise would involve computations in higher dimensional space.

Mathematical definition: K(x, y) = <f(x), f(y)>. Here K is the kernel function, x, y are n dimensional inputs. f is a map from n-dimension to m-dimension space. < x,y> denotes the dot product. usually m is much larger than n.

Intuition: normally calculating <f(x), f(y)> requires us to calculate f(x), f(y) first, and then do the dot product. These two computation steps can be quite expensive as they involve manipulations in m dimensional space, where m can be a large number. But after all the trouble of going to the high dimensional space, the result of the dot product is really a scalar: we come back to one-dimensional space again! Now, the question we have is: do we really need to go through all the trouble to get this one number? do we really have to go to the m-dimensional space? The answer is no, if you find a clever kernel.

4| Agoritmi

SVM (Support Vector Machines)

Explicație oferită de Lili Jiang, Quora Data Science Manager

Simple Example: x = (x1, x2, x3); y = (y1, y2, y3). Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3), the kernel is K(x, y) = (<x, y>)^2.

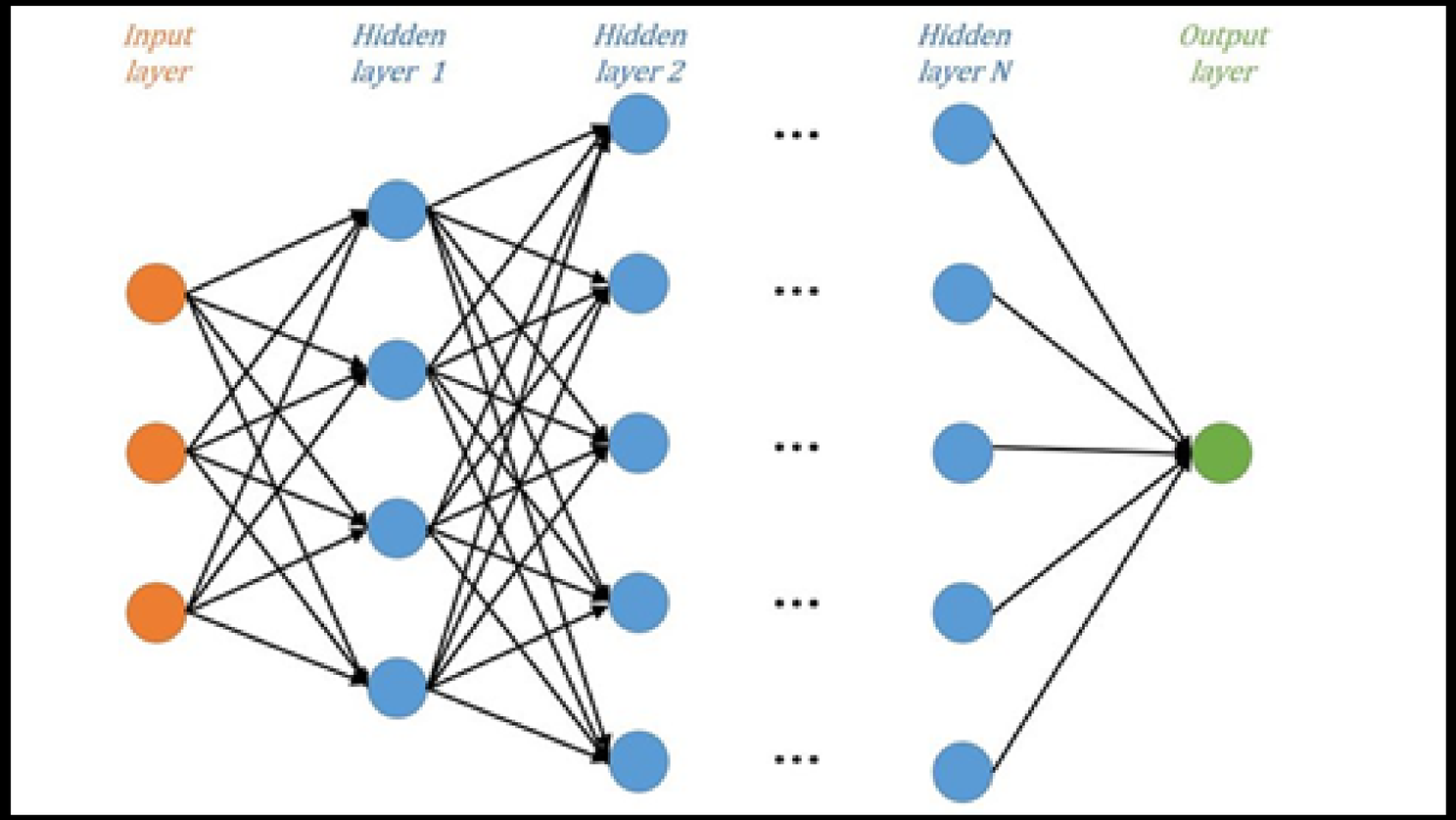
Let's plug in some numbers to make this more intuitive: suppose x = (1, 2, 3); y = (4, 5, 6). Then:
f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)
f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)
<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

A lot of algebra. Mainly because f is a mapping from 3-dimensional to 9 dimensional space.

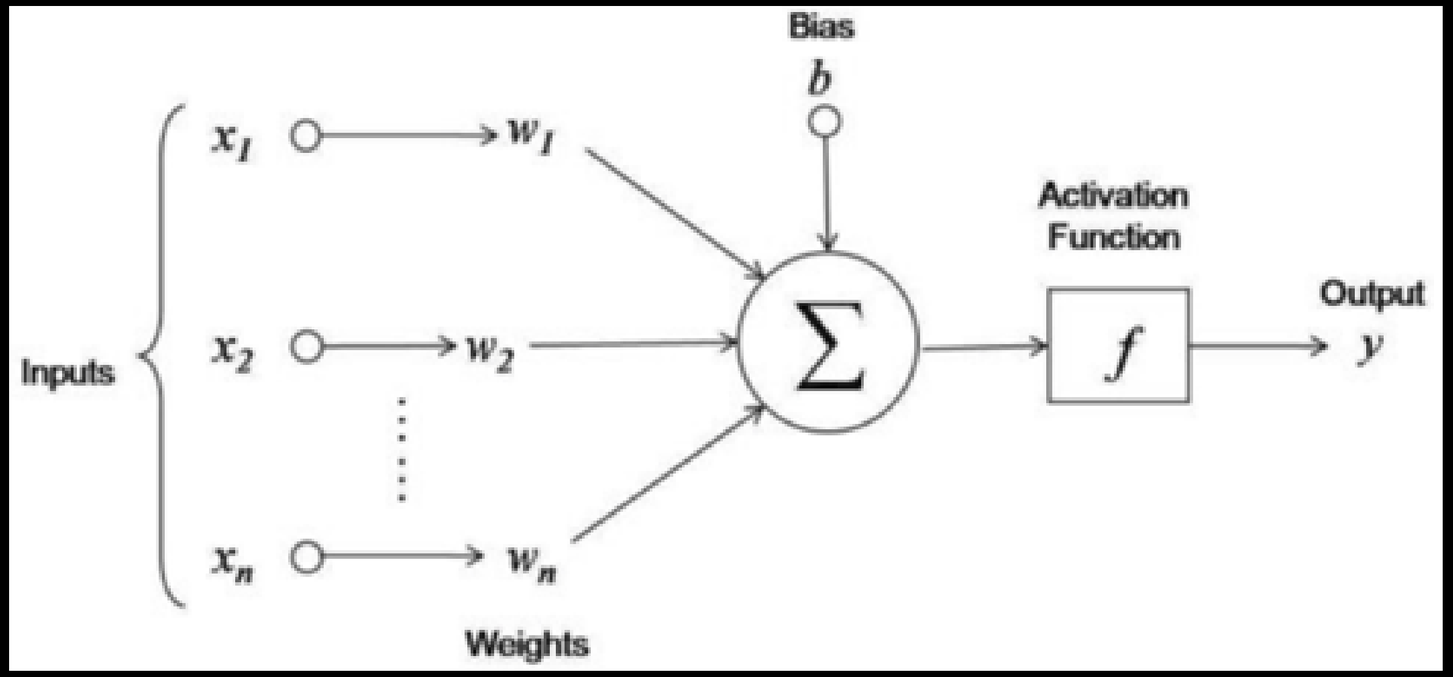
Now let us use the kernel instead:
K(x, y) = (4 + 10 + 18) ^2 = 32^2 = 1024
Same result, but this calculation is so much easier.

Rețele neuronale artificiale *feedforward*

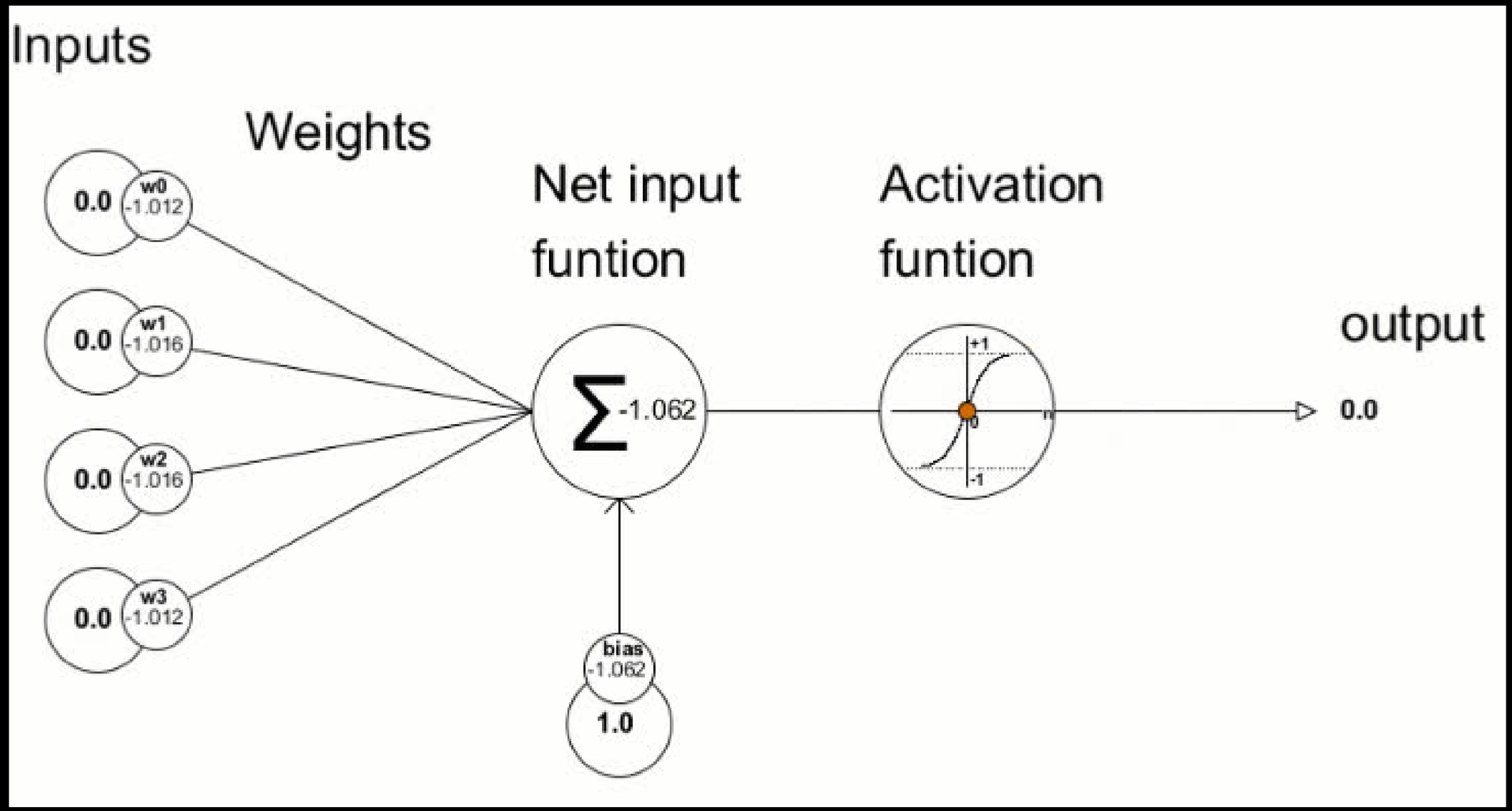
- 1 nivel de intrare (inputuri)
- n nivele ascunse cu număr variabil de neuroni
- 1 nivel de ieșire (1 neuron în cazul nostru, pentru că prezicem 1 singur rezultat)



Ce avem într-un neuron?



4| Algoritmi


Inputs
Weights
Net input
funtion
Activation
funtion
output
0.0
w0
-1.012
0.0
w1
-1.016
0.0
w2
-1.016
0.0
w3
-1.012
-1.062
bias
-1.062
1.0
+1
0
-1
0.0

Rețele neuronale artificiale *feedforward*

- **Funcții de activare uzuale: ReLU, sigmoid**

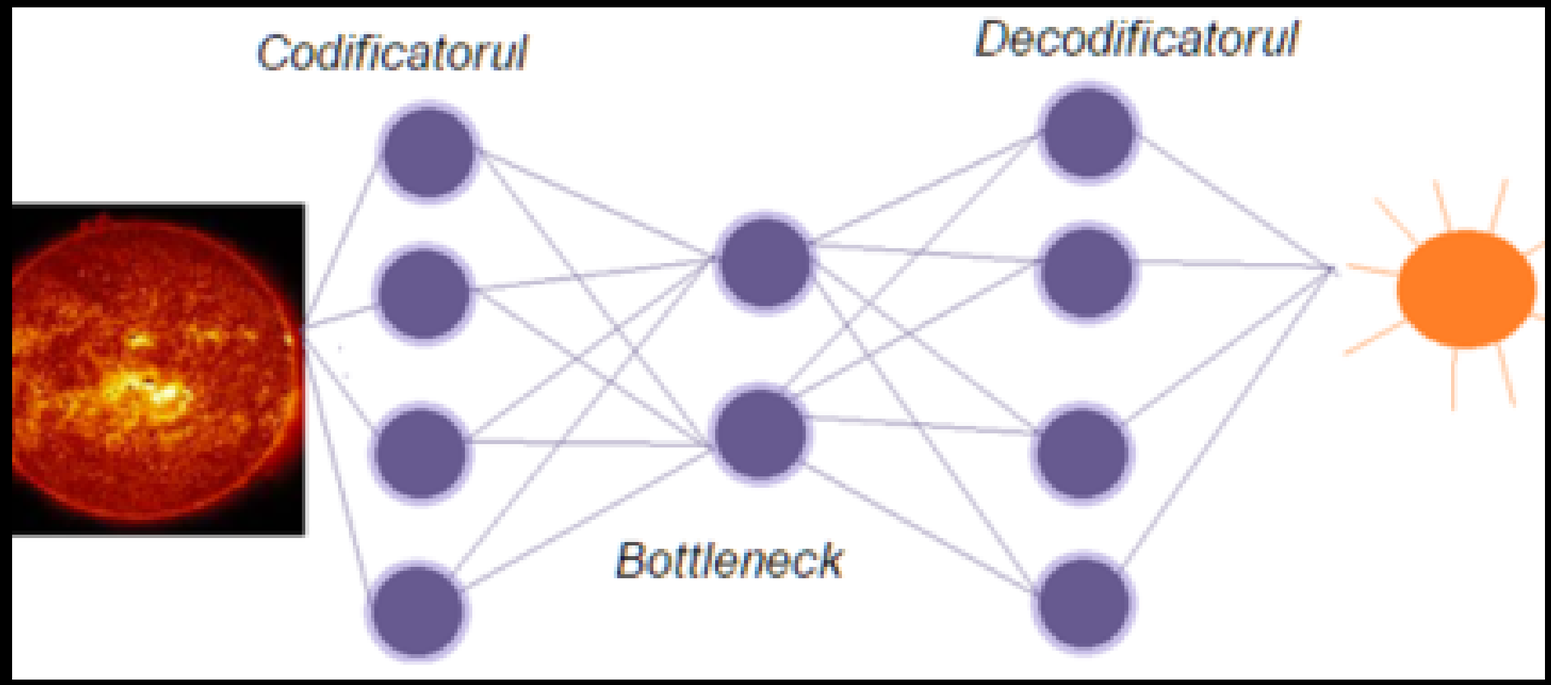
$$ReLU(z) = \max(0, z)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Găsirea arhitecturii unei rețelei = găsirea hiperparametrilor optimi**
  (hiperparametri = parametri ce nu sunt învățați; e.g. numărul de nivele ascunse, numărul de neuroni, numărul de exemple după care au loc modificările - *batch size* - , rata de învățare)

Autocodificatoarele (*Autoencoders*)

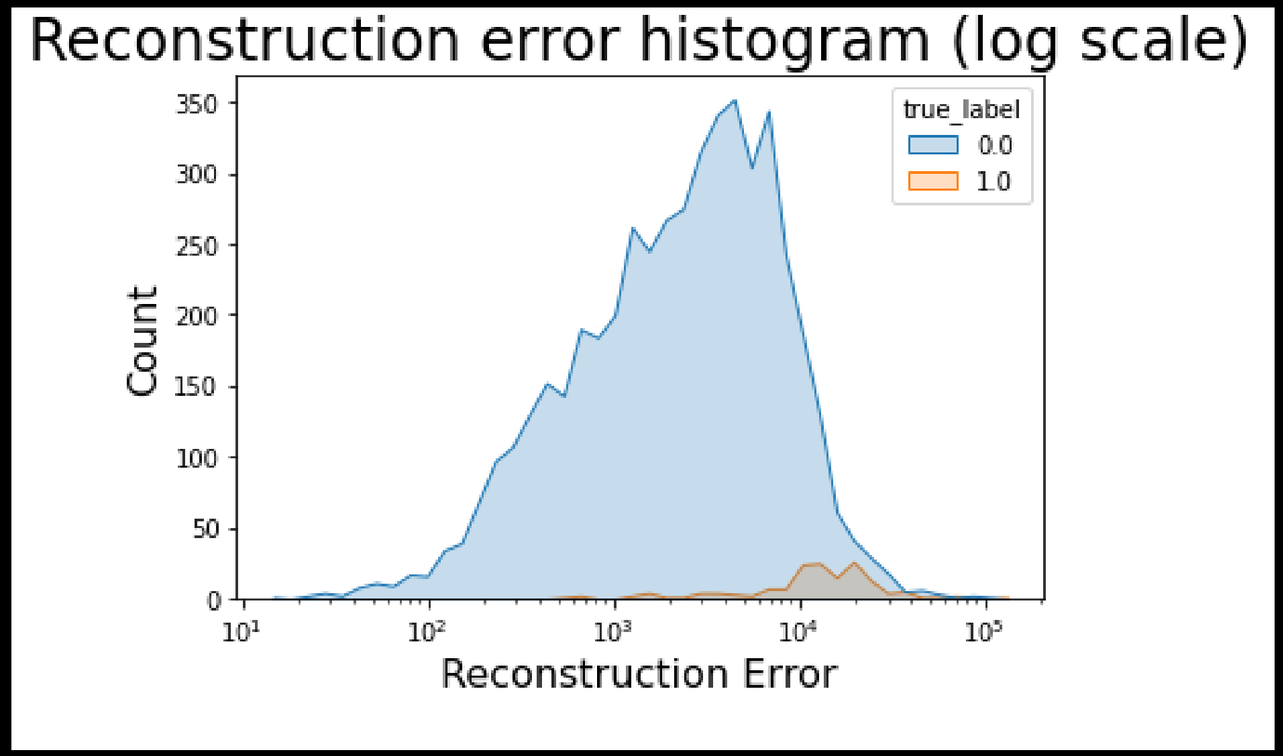
rețele neuronale speciale, simetrice, #intrări = #ieșiri, care învață să **reproducă** într-un mod **simplificat** datele de intrare



Autocodificatoarele (*Autoencoders*)

pentru

detecția anomaliilor pe baza erorii de reconstrucție



Optimizări

PONDERI PENTRU ECHILIBRAREA CLASELOR

Oferă **mai multă importanță** exemplelor din clasa minoritară - adică **”penalizează” greșelile mai tare** la antrenare --> face ca **schimbările** aduse coeficienților să fie **mai semnificative**

Necesar pentru a identifica o proporție semnificativă de evenimente minoritare

Optimizări

PONDEREA UNUI EVENIMENT POTENȚIAL GEOEFECTIV = 1/2 * PONDEREA UNUI EVENIMENT GEOEFECTIV

Am considerat CME-urile observate cu până la 5 zile înainte de furtunile fără o cauză determinate ca fiind potențial geoefective

Schimbarea semnificativă a fost o creștere de 2% în precizia rețelei neuronale, dar cu prețul unei scăderi de ~6% pentru hit rate (sensibilitate)

Optimizări

UTILIZAREA ALTUI PRAG ÎN LOCUL CLASICULUI 0.5

Comparație pentru rețeaua neuronală

	0.5	0.9
Hit rate:	90.8%	72.9%
Precizie:	8.3%	15.7%

Optimizări

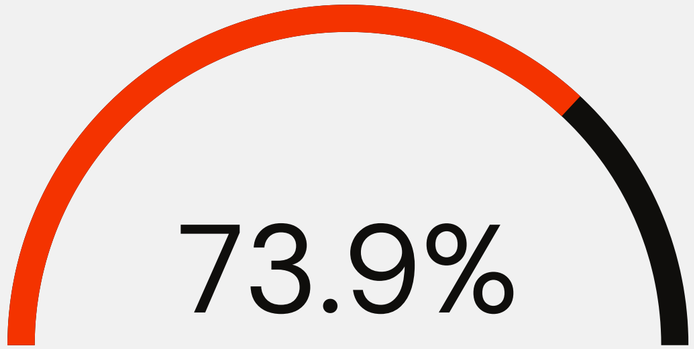
SMOTE
SYNTHETIC
MINORITY
OVERSAMPLING
TECHNIQUE

Se alege aleatoriu 1 punct din clasa minoritară
Se aleg K cei mai apropiați vecini
Se alege unul dintre vecini aleatoriu
Se creează un punct nou aleatoriu pe dreapta între cele 2

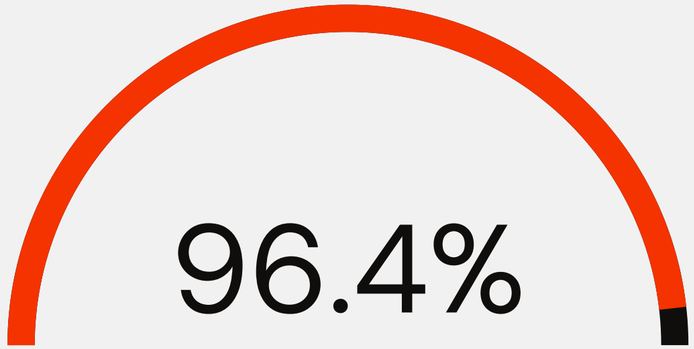
Rezultatele la antrenare s-au îmbunătățit considerabil, dar la testare pe setul de date originale, rezultatele sunt asemănătoare

Modelul cu cele mai bune rezultate de până acum

ANN cu 1 nivel ascuns,
1 neuron
learning rate = 1e-3
batch size = 32
epochs = 40
funcții de activare: ReLU & sigmoid (0.9 threshold)



Sensibilitate (hit rate)



Specificitate

96.7%
Acuratețea

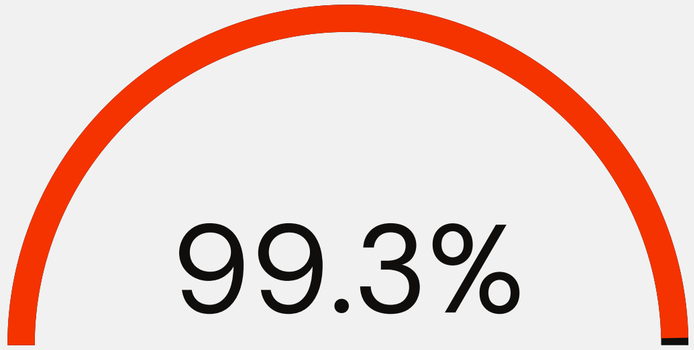
73.9%
Rata de real pozitive

96.4%
Rata de real negative

14.7%
Precizia

Modelul cu cea mai bună precizie de până acum

KNN cu:
5 vecini
ponderi uniforme
distanța Minkowski



Specificitate

Sensibilitate

99.2%
Acuratețea

2.8%
Rata de real pozitive

99.3%
Rata de real negative

56.6%
Precizie

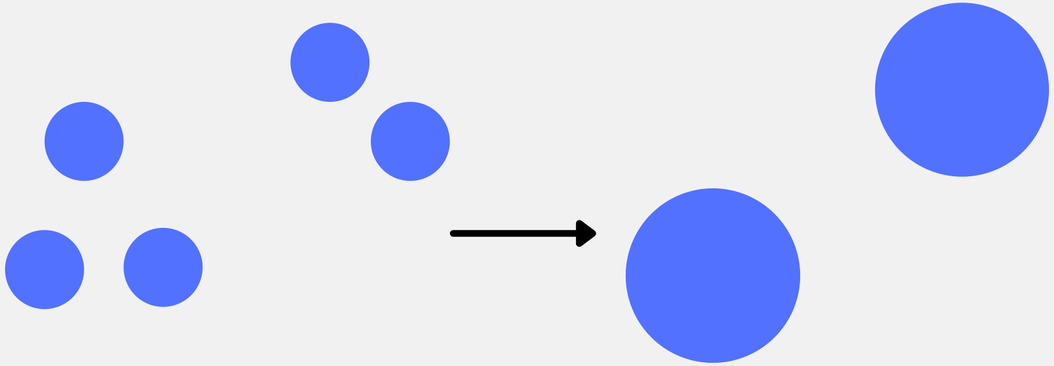
Comparație din punct de vedere al proporției furtunilor identificate & a preciziei

pentru cele mai bune variante pentru fiecare model, cu diverse optimizări

98.4%	7.5%	78.7%	73.9%	85.8%
Regresia Logistică	KNN	SVM	ANN	Autoencoder
3.2%	33.3%	10.5%	14.7%	9%
Regresia Logistică	KNN	SVM	ANN	Autoencoder

Vizualizările predicțiilor

UMAP - tehnică eficientă de reducere a numărului de dimensiuni



Legenda culorilor

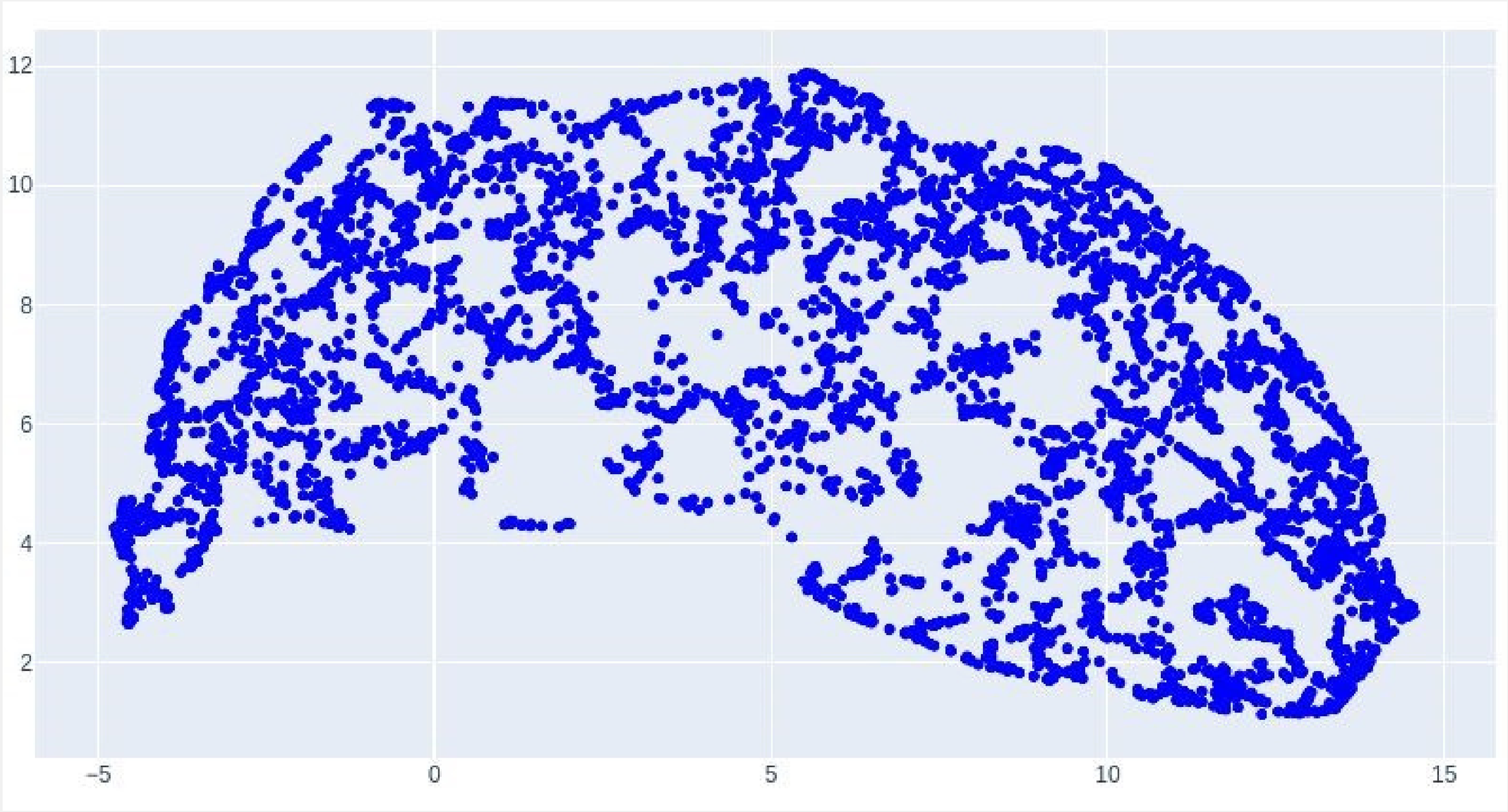
TN (albastru) = Real Negative

FP (galben) = Fals Pozitive

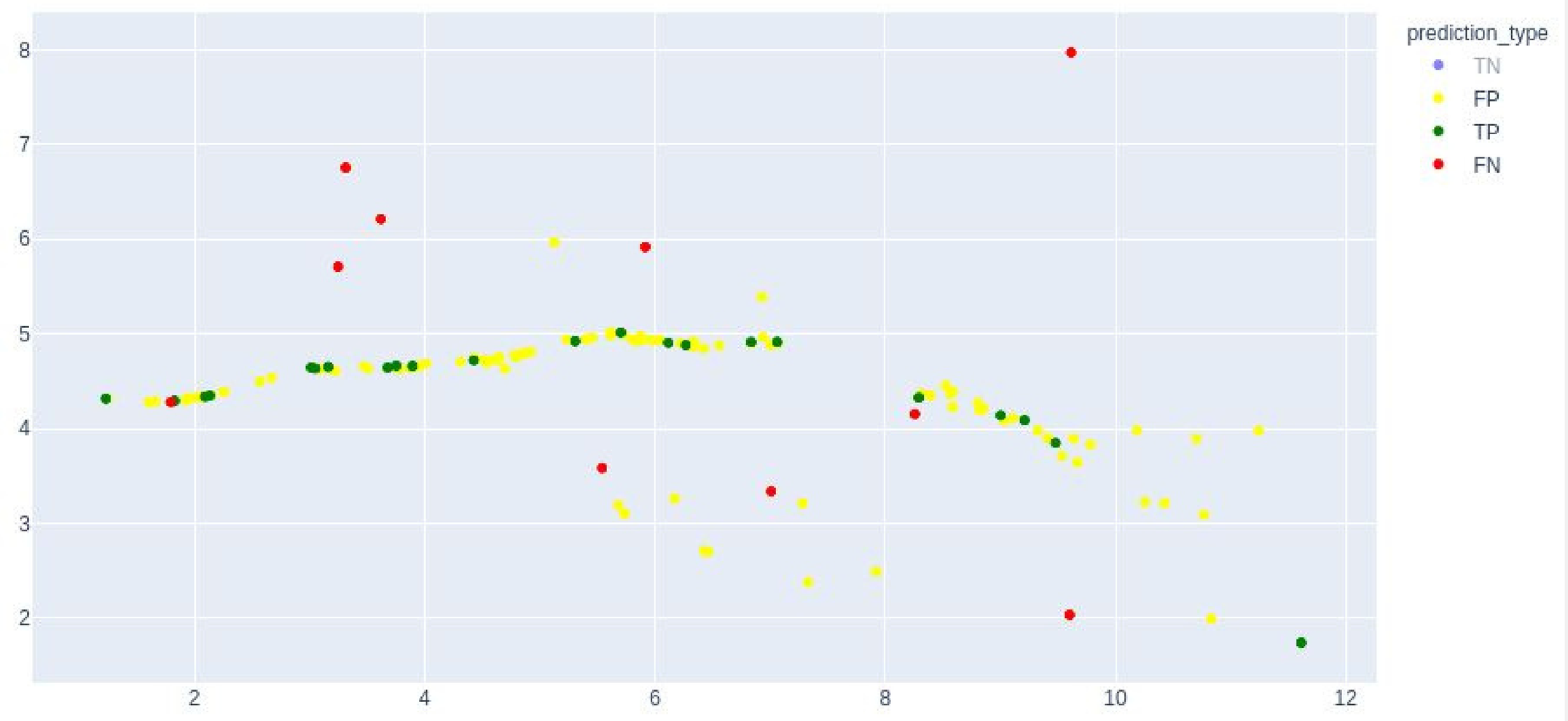
TP (verde) = Real Pozitive

FN (roșu) = Fals Negative

Real Negative



Fals & Real Pozitive, Fals Negative



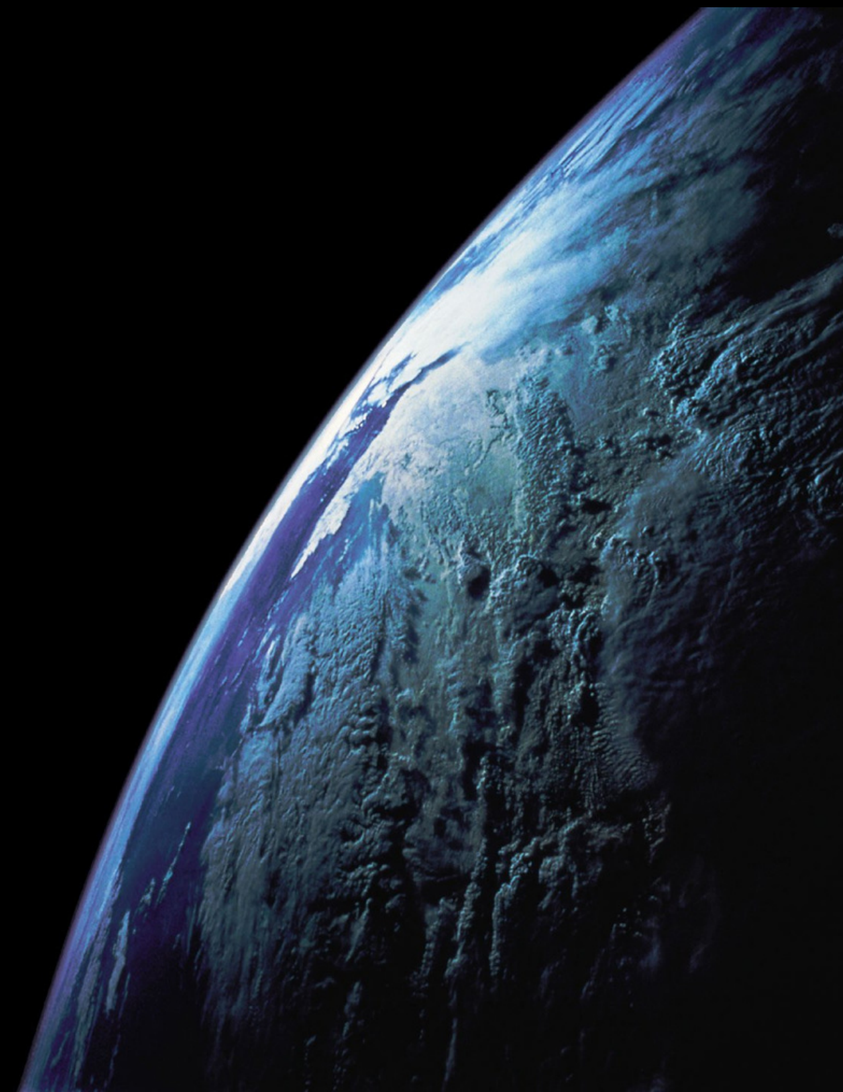
Privind rezultatele dintr-o perspectivă solară

Sensibilitatea este cea mai importantă măsură a performanței, datorită costului asociat unui fals negativ (e de preferat o alarmă falsă în detrimentul unei potențiale furtuni neidentificate)
Majoritatea alarmelor false provin de la un tip de CMEs, denumit halo, care este considerat, în general, cu cel mai mare potențial geoefectiv.
Dintr-o perspectivă fizică, acest lucru era de așteptat.

Îmbunătățiri ulterioare

Încercarea îmbunătățirii rezultatelor cu un model de tip ansamblu

Studierea rezultatelor după adăugarea unei/unor informatii privind poziția originii ejecției




Mulțumesc pentru atenție!

Vreau mai multe informații
De unde pot începe?

1. Scikit-learn

(bibliotecă ce conține majoritatea implementărilor necesare de bază pentru machine learning + exemple)

2. TensorFlow + Keras

(frameworkuri pentru rețele neuronale și deep learning + multe resurse explicative)

3. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, A.Géron

(carte cu foarte multe exemple practice și explicații simple)

Vreau mai multe informații
De unde pot începe?

4. UMAP

(Documentația oficială: https://umap-learn.readthedocs.io/en/latest/)

5. Email-ul meu: andreeapricopi@gmail.com

(Orice întrebări legate de acest proiect sau despre *machine learning* în general)