Brain anomaly detection

Tănase Victor-Flavian

Grupa 244

**Descrierea problemei**

Se pune la dispozitie un set CT scan-uri ale creierului : imagini in tonuri de gri de dimensiune 224x224 pixeli, fiecare exemplu aflandu-se intr-una din clasele 0(pentru CT scan-urile normale) si 1(pentru cele ce contin anomalii).

**Scopul problemei**

Se doreste gasirea unui clasificator care poate distinge dintre un CT scan normal si unul anormal, cu performanta cea mai buna avand urmatoarea partitionare a imaginilor : setul de antrenare contine 15000 de exemple etichetate, cel de validare 2000 de exemple etichetate, iar setul de testare contine alte 5149 de imagini neetichetate. Metoda de evaluare folosita va fi [media F1](https://en.wikipedia.org/wiki/F-score).

**Clasificatorul Multinomial Naïve Bayes**

Preprocesarea datelor :

* Imaginile reprezinta fisiere in format png cu dimensiune 224x224 pixeli
* Pentru manipularea imaginilor am folosit libraria PIL(python image library)
* Cum imaginile sunt greyscaled, le putem pune in numpy array-uri de dimensiune 1 x 224 \* 224
* O mentiune foarte importanta este faptul ca la citire, trebuie sa fie garantat ca pozele sunt procesate in ordine pentru antrenarea corecta a modelului.

Utilizarea propriu-zisa a modelului:

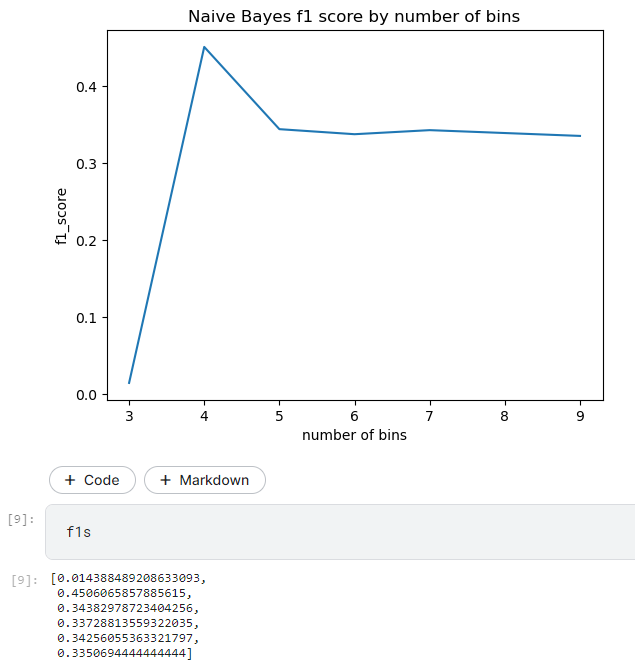
**Prima varianta -> utilizarea de binuri**

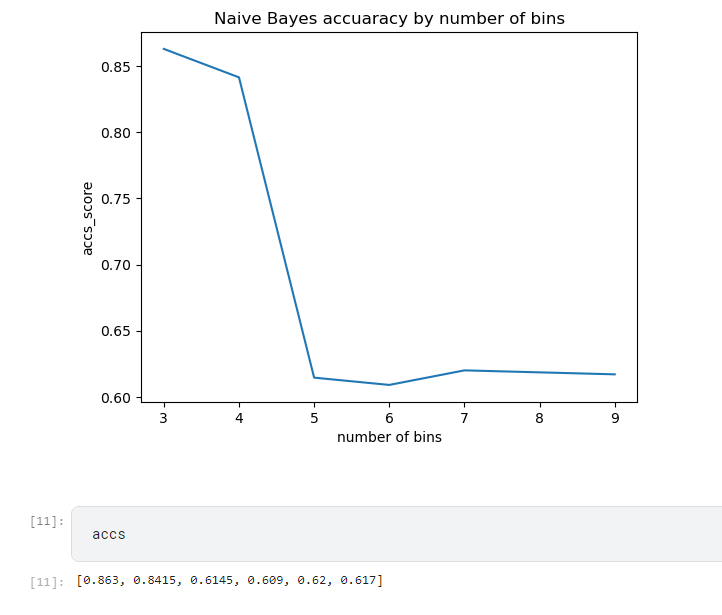
Un pixel poate lua valori intre 0-255. Functia createBins(n) va imparti intervalul 0-255 in n-1 subintervale, generand capetele acestora. Deci exista n - 1 clase in care se vor putea incadra imaginile. In final, clasa cu probabilitatea cea mai mare va decide alegerea pe care clasificatorul o va face.

Plecand de la aceasta ipoteza, s-a decis alegerea catorva valori ale lui n pentru care se imparte intervalul. Deoarece setul de antrenare este unul destul de mare comparativ cu memoria RAM, train-ul nu s-a putut face pe toate datele deodata. Solutia la aceasta problema a fost functia partial\_fit a modelului. Parametrii functiei sunt : X, y, classes, sample\_weight(default=None).

* **X** : reprezinta multimea imaginilor, in cazul prezentat X primeste chunk-ul de imagini
* **y** : label-urile respective imaginilor, in cazul prezentat y primeste chunk-ul de labeluri.
* **classes\_sample** : reprezinta lista tuturor claselor care pot aparea in y, 0 si 1.
* **sample\_weight** : bias in selectia claselor, nu a fost folosit deoarece, nu s-a dorit adaugarea unui bias in predictia imaginilor.

Dupa incadrarea fiecarei imagini in clasa corespunzatoare si efecutarea predictiei pe testele de validare s-a calculat acuratetea si scorul f1 verificand rezultatul obtinut cu valorile din validation\_labels. O vizualizare a acestei simulari poate fi vazuta in graficele de mai jos.





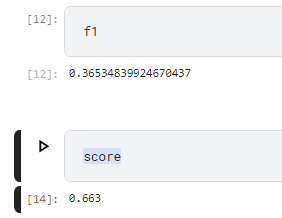
In simularea facuta, acuratetea maxima este data de impartirea intervalului in 2 subintervale, urmate de 3 subintervale.

Insa in graficul pentru f1 se poate observa ca pentru 3 binuri, scorul este foarte mic in comparatie cu celelalte. In cele din urma am ales 4 binuri pentru a antrena clasificatorul si in final al testa pe datele de test.

***SCOR PE PLATFORMA : 0.44363***

**A doua varianta -> antrenare normala**

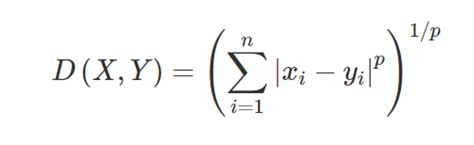
Cu aceeasi preprocesare a datelor si aceeasi antrenare partiala, s-a dorit observarea rezultatelor si cu o antrenare clasica a modelului.



Dupa cum se poate observa, scorul este unul mai scazut decat in varianta precedenta, dar si acuratetea este mai scazuta. Acest lucru confirma ca acest clasificator are rezultate demne de luat in considerare, dar nu optime pentru problema prezentata.

**Clasificatorul KNN**

Clasificatorul KNN(K-Nearest Neighbours) este un clasificator simplu, dar puternic in machine learning folosit in task-uri de clasificare si regresie. Algoritmul functioneaza in modul urmator: acesta identifica cele K puncte (K dat) cele mai apropiate de fata de un anumit punct de date. In aflarea acestor puncte se foloseste distanta dintre puncte, in cazul problemei noastre, distanta Minkowski, care este parametrul default, calculata dupa formula:

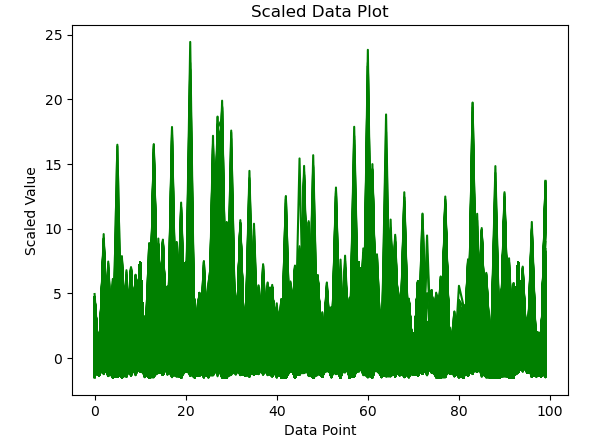


Pentru antrenarea modelului, s-a folosit aceeasi preprocesare a detelor ca in cazul anterior. Clasificatorul KNN din sklearn foloseste mai multi parametri, dintre care cei mai importanti : **n\_neighbors, weights, algorithm**, **p : int*, default=2.***

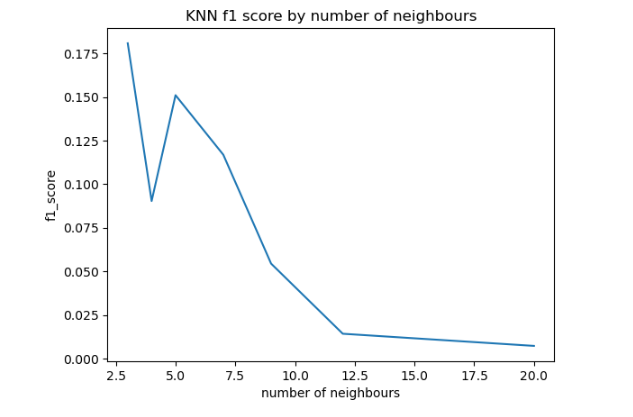
* **n\_neighbours** : reprezinta K-ul descris mai devreme. Discutia se va purta in jurul acestui parametru, in principal.
* **weights** : reprezinta bias-ul catre anumite puncte, ceea ce influenteaza distanta la care se afla. In antrenarea modelului s-a folosit parametrul default, “uniform”, in care toate punctele sunt “egale”, adica distanta este cea efectiva.
* **alogrithm** : este folosit pentru alegerea algoritmului care efectueaza operatiile calcularii distantei, by default initializat cu “auto”, care lasa clasificatorul sa-si aleaga singur metoda care i se pare cea mai buna.
* **p** : puterea din calcularea distantei minkowski. Pentru p=1 -> distanta Manhattan, p=2 -> distanta Euclidiana. A fost pastrata varianta default, adica p=2.

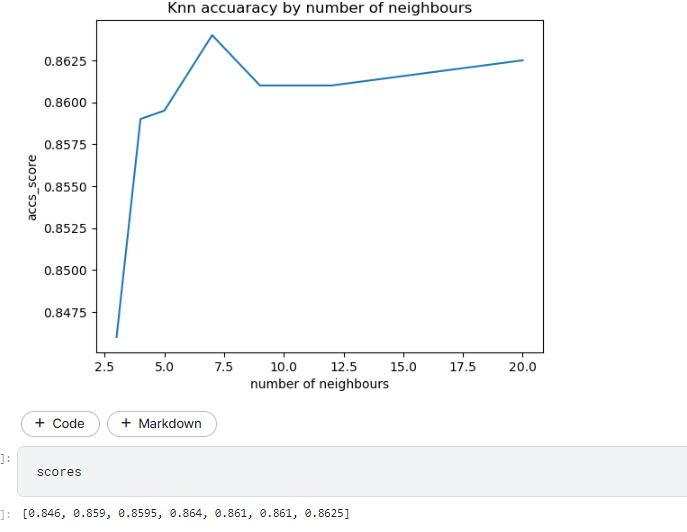
Problema in cazul acestui clasificator, s-a redus la gasirea numarului optim de vecini.

Fata de preprocesarea datelor pentru clasificatorul NB, s-a adaugat si ***normalizarea***. Pentru normalizare, s-a folosit StandardScaler() din sklearn. Acesta transforma datele in asa fel incat media lor sa fie 0 si deviatia standard -1. Transformarea ne garanteaza ca valorile datelor, in cazul nostru pixelilor, vor fi in jurul valorii 0. Acest lucru reduce semnificativ costul de computatie



Pentru alegerea optima a numarului de vecini s-au ales valori crescatoare ale numarului de vecini si, iterativ, s-a creat si antrenat cate un model cu numarul de vecini respectiv. Dupa fiecare antrenare, fiecare model a fost evaluat, cu aceleasi metode ca in cazul celorlalte modele. Rezultatele pot fi vazute mai jos.





In urma evaluarii, se poate observa ca acest clasificator nu a fost nici pe aproape de performantele clasificatorului Naïve Bayes, asa ca s-a decis abandonarea lui si nu a fost evaluat pe datele de test.

**Clasificatorul Random Forest**

Clasificatorul Radom Forest este un algoritm popular de machine learning uitilizat pentru task-uri de clasificare. La baza acestuia sta o tehnica numita *ensemble learning* care presupune antrenarea mai multor clasificatoare si alegerea predictiei finle in urma agregarii rezultatelor acestora. Un exemplu concret de ensemble learning este antrenarea mai multor [arobri de decizie](https://www.ibm.com/topics/decision-trees) pe submultimi alese aleator din multimea de training. In urma predictiilor arborilor, clasificatorul alege votul majoritar. Aceasta asamblare de arbori de decizie reprezinta un clasificator Random Forest.

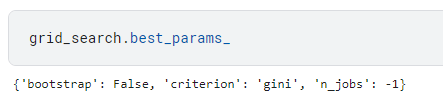
In antrenarea modelului s-au folosit imaginile incarcate in numpy array-uri, de aceasta data fara sa le normalizam, deoarece poate duce la pierdere de informatie in cadrul clasificatorului si ar putea sa-i reduca performanta.

Hyperparametrii care au fost alesi pentru testare sunt: **n\_estimators, criterion si bootstrap, n\_jobs**

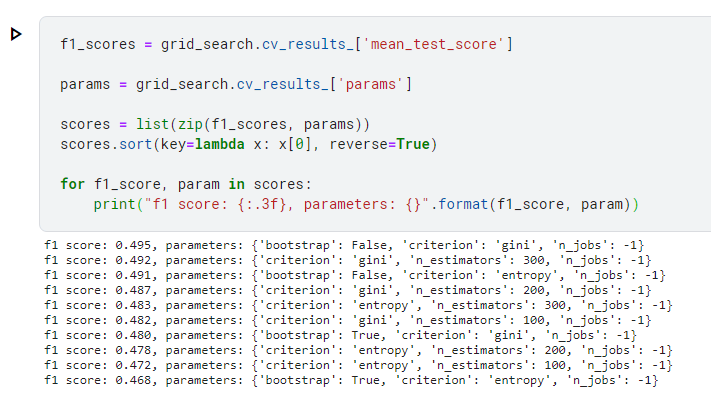
* **n\_estimators :** numarul de arbori de decizie din “padure”
* **criterion :** functia care masoara calitatea split-ului la construirea arborilor de decizie
* **bootstrap:** tehnica statistica, folosita pentru a minimiza varianta si a crea arbori cat mai diferiti, ceea ce previne overfitting-ul
* **n\_jobs :** parametru care gestioneaza resursele de procesare, in general setat cu -1 pentru a folosi toate resursele disponibile in timpul computatiei

Pentru a testa diverse combinatii intre acesti parametri, s-a folosit [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) din skelearn. Pentru a utiliza acest tool, avem nevoie de o lista de dictionare cu parametri ca si chei si valorile pe care le va lua parametrul cand apelam modelul ca si valori ale dictionarului. In principiu, modelul va fi antrenat cu produsul cartezian al parametrilor din fiecare dictionar. In functia GridSearchCV s-a folosit si cross validation pentru evaluarea performantei in timpul antrenarii. Acest lucru ajuta la prevenirea overfitting-ului.

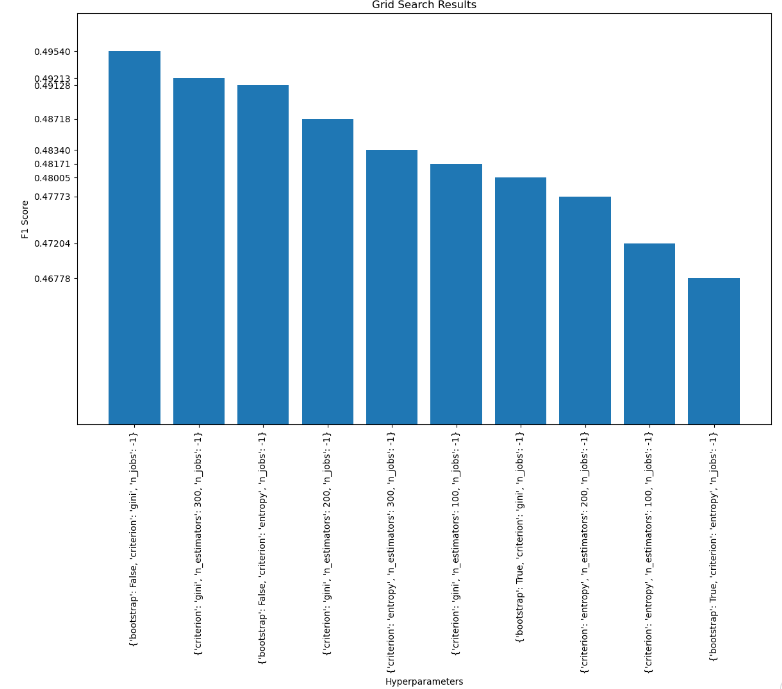
In urma antrenarii modelelor, cu functia best\_params\_ obtinem parametri modelului cu performanta cea mai buna din timpul antrenarii.

****

Cu toate ca acest model a avut cea mai buna performanta, in antrenarea lui lipsesc parametrii din primul dictionar folosit. De aceea sa analizeaza si performanta celorlalte modele. Se extrag parametrii si scorurile f1 din rezultatele din gridSearch cu functia cv\_results\_.

****

Reprezenatrea grafica :

****

In urma analizarii rezultatelor, s-a decis utilizarea parametrilor : bootstrap=False, n\_estimators = 500, criterion = "gini", n\_jobs=-1

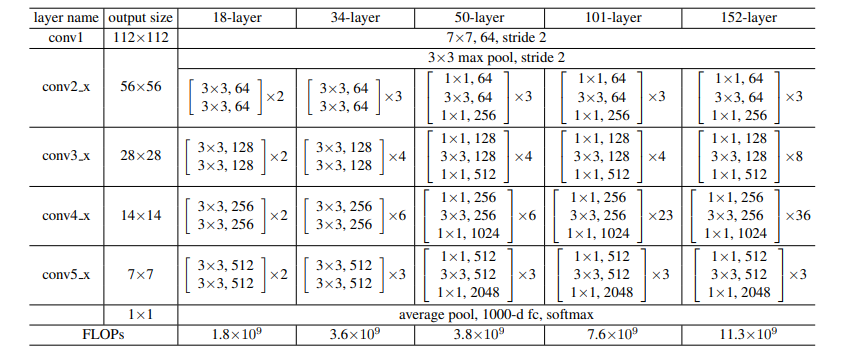
***SCOR PE PLATFORMA :*** 0.52818

Nota: cu alte modele ale aceluiasi clasificator s-au obtinut scorurile : 0.52146, 0.52054.

**Retele Neuronale Convolutionale(CNN)**

Retelele neuronale convolutionale sunt un tip de retea neuronala deep learning, folosita la analiza imaginilor si videoclipurilor. Caracteristica principala a CNN-urilor este uitlizarea straturilor convolutionale, care aplica filtre(numite kernel-uri) la imaginea de intrare pentru a extrage caracteristici precum marginile, formele si texturile. Acestea sunt combinate mai apoi in straturile ulterioare pentru a forma reprezentari mai complexe ale imaginii.

Pentru aceasta problema s-a ales arhitectura Residual Network 18(ResNet18) descrisa in amanunt in [acest paper publicat in 2015](https://arxiv.org/pdf/1512.03385.pdf). La baza arhitecturii, stau utilizarea de shortcut-uri in retea si utilizarea unor blocuri de straturi convolutionale.



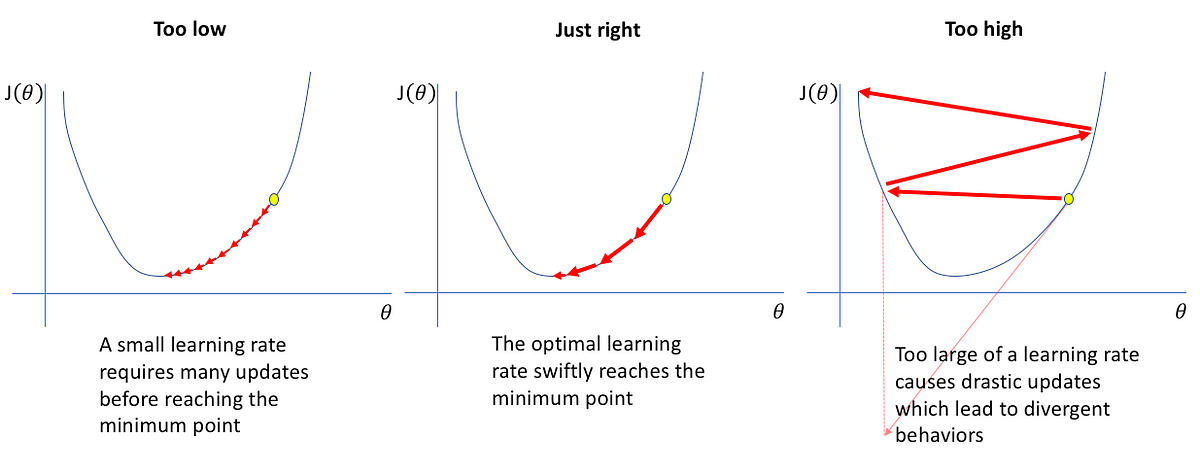
Distributia straturilor convolutionale

In acest proiect, s-a folosit implementarea arhitecturii din [torchvision](https://pytorch.org/vision/main/_modules/torchvision/models/resnet.html#ResNet18_Weights) la care s-au adus mici modificari pentru a putea folosi imaginile grayscaled din setul de date. Chiar daca exista o versiune pre-antrenata a modelului, aceasta nu s-a folosit.

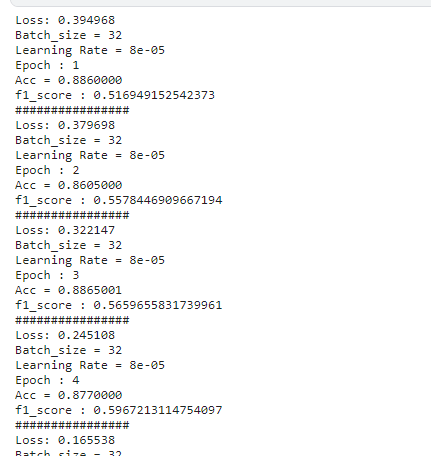
Parametrii folositi pentru straturile retelei: **input\_channels, output\_channels, kernel\_size, stride, padding, bias**.

* **input\_channels :** numarul de canale in inputul tensorului. Tensorul reprezinta o imagine, in care fiecare canal corespunde unei caracteristici diferite ale inputului, in cazul problemei prezentate, va fi 1, in general 3 pentru valori RGB ale pixelilor
* **output\_channels :** reprezinta numarul de mapari ale featurerilor care sunt produse de stratul convolutional
* **kernel\_size :** dimensiunea kernelului folosit. Kernelul este generat cu valori aleatorii si efectueaza operatii de convolutie pe suprafata imaginii de input pentru a detecta feature-uri, pattern-uri.
* **stride :** pasul pe care il face kernelul in timp ce se plimba prin tensor
* **padding :** adaugarea de margini extra de pixeli tensorului de input inaintea aplicarea stratului convolutional, pentru a pastra dimensiunea spatiala a inputului si de a se asigura ca feature map-urile din output au aceeasi marime ca inputul
* **bias :** terman adaugat la produsul dintre valorile din filtru si valorile pixelilor din input. In cazul problemei va fi False

Pentru antrenarea retelei, s-au folosit foarte multe variatii ale hyperparametrilor:

* **num\_epochs** : numarul de epoci pentru care este antrenata reteaua
* **batch**\_**size** : marimea chunk-urilor de date care intra in retea, asignat la crearea dataloaderlor
* **lr :** learning rate-ul -> marimea pasului pe care optimizer-ul il face cand ajusteaza weight-urile din retea in timpul antrenarii

In functia trial\_and\_error, am luat pe rand mai multe combinatii de parametri si pentru fiecare model, dupa fiecare epoca am testat performanta retelei si am salvat parametrii.



Secvente din timpul testarii

In cele din urma s-a ales un model cu parametrii “optimi” pentru modalitatea de testare aleasa. Problema principala a fost ca rezultate nu au putut fi reproduse, din cauza ca nu s-a luat in considerare salvarea random\_seed-ului, astfel kernelul neputand fi recreat, acesta fiind generat aleator. Solutia finala a fost antrenarea retelelor cu parametrii cei mai buni si testarea pe datele de test in cazul in care rezultatele au fost satisfacatoare.