MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student	Cigan Oliviu-David	GRUPA:	30132	Nota	

Brain Tumor Detection System

Autor: Cigan Oliviu-David

Grupa**: 30132**

AN UNIVERSITAR: 2022-2023

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Cuprins

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

1. Scopul Proiectului

Brain-Tumor-Detection-System este un proiect inovator in industria medicala care are ca scop imbunatatirea procesului de diagnosticare a tumorilor cerebrale. Prin colectarea si analizarea imaginilor de tip radiografie a creierului, acest sistem ajuta medicii sa determine cu aproximatie prezenta unei tumori.

Acest proiect este extrem de important deoarece diagnosticarea timpurie a tumorilor cerebrale poate face diferenta dintre viata si moarte pentru pacienti. Cu cat o tumora este descoperita mai devreme, cu atat sansele de a trata si vindeca pacientul sunt mai mari.

Brain-Tumor-Detection-System utilizeaza tehnologie de ultima ora pentru a procesa si analiza imaginile de radiografie a creierului, oferind astfel un nivel ridicat de precizie si fiabilitate. Sistemul este proiectat sa identifice anomalii in imaginile radiologice, astfel incat sa poata fi identificate zonele de interes si sa se ofere o evaluare precisa.

In plus, Brain-Tumor-Detection-System poate fi utilizat in timp real, ceea ce inseamna ca medicii pot avea acces instantaneu la rezultatele analizelor, permitand astfel diagnosticarea rapida si precisa. Acest sistem poate contribui semnificativ la reducerea timpului necesar pentru diagnosticarea tumorilor cerebrale, ceea ce poate avea un impact major asupra sanatatii si bunastarii pacientilor..

a. Obiective

Obiectul acestui proiect, este sa ofere medicilor o unealta utila pentru a intelege si analiza mai usor radiografiile la creier, contribuind astfel la imbunatatirea procesului de diagnosticare a tumorilor cerebrale.

Este important de mentionat ca acest proiect nu are ca scop inlocuirea medicilor sau analizatorilor de radiografii, ci sa fie un suport pentru acestia. Sistemul este proiectat sa ofere informatii suplimentare si evaluari preliminare ale imaginilor radiologice, astfel incat medicii sa poata lua decizii mai informate si sa ofere un diagnostic mai precis.

Brain-Tumor-Detection-System foloseste tehnologie avansata pentru a procesa imaginile radiologice ale creierului, identificand anomalii si zone de interes. In plus, sistemul poate fi utilizat in timp real, astfel incat medicii sa poata accesa informatiile

necesare imediat, ajutand astfel la reducerea timpului necesar pentru diagnosticarea tumorilor cerebrale.

Este important sa subliniem faptul ca acest proiect este conceput pentru a fi o unealta utila pentru medici, oferindu-le un suport suplimentar si ajutandu-i sa ia decizii mai informate in procesul de diagnosticare a tumorilor cerebrale. Nu are ca scop inlocuirea lor sau a analizatorilor de radiografii, ci sa fie un instrument de sprijin pentru a imbunatati procesul de diagnosticare si tratament pentru pacienti.

b. Specificații

In cadrul acestui proiect, s-a dezvoltat un sistem automat pentru colectarea imaginilor si diagnosticarea tumorilor cerebrale. In prima etapa a procesului, se face colectarea automata a imaginilor de radiografii ale creierului, care sunt redimensionate la o dimensiune standard de 300px x 300px pentru a putea fi prelucrate mai eficient.

In etapa urmatoare, imaginile sunt prelucrate intr-un format numeric, prin intermediul unui proces de preprocesare. Acest proces are ca scop eliminarea zgomotului si a altor artefacte din imagine, pentru a permite o analiza mai precisa a datelor.

Odata prelucrate, imaginile sunt trecute prin trei algoritmi diferiti de machine learning: LogisticRegression, SVM si RandomForestClassifier. Fiecare dintre aceste algoritmi este evaluat si comparat pentru a determina care dintre ele ofera cea mai buna acuratete in diagnosticarea tumorilor cerebrale.

Dupa ce algoritmul de machine learning a fost selectat, sistemul poate analiza trei tipuri diferite de tumori cerebrale: pituitary_tumor, meningioma_tumor si glioma_tumor. In functie de datele colectate si analizate, sistemul poate oferi o concluzie cu privire la prezenta sau absenta cancerului. De asemenea, performanta sistemului este destul de inalta, chiar si luand in considerare dataset-ul destul de complex utilizat in acest proiect. Sistemul raspunde foarte rapid si eficient la cerintele utilizatorilor, oferind o acuratete buna in diagosticarea tumorilor cerebrale.

In final, sistemul poate afisa o vizualizare a imaginii, impreuna cu concluzia oferita de sistem. Aceasta poate fi deosebit de utila pentru medicii care se ocupa de diagnosticarea tumorilor cerebrale, oferindu-le o metoda precisa si rapida pentru a determina daca pacientii lor sunt sau nu afectati de aceasta afectiune grava.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Acest sistem de detectare a tumorilor cerebrale a fost creat folosind Python si Jupyter Notebook. Pentru a dezvolta acest sistem, s-au utilizat o serie de librarii importante, inclusiv Sklearn, numpy, pandas si matplotlib, care au fost esentiale in procesul de colectare si preprocesare a datelor.

2. Studiu bibliographic

Bibliografia mea include următoarele cărți de referință în domeniul deep learning și prelucrarea imaginilor digitale:

- Chollet, F. (2018). Deep Learning with Python. Shelter Island, NY: Manning Publications Co.

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. Cambridge, MA: MIT Press.

- Solomon, C. (2010). Fundamentals of Digital Image Processing. Hoboken, NJ: John Wiley & Sons, Inc.

- Patterson, J., & Gibson, A. (2017). Deep Learning - A Practitioner's Approach. Sebastopol, CA: O'Reilly Media, Inc.

- Geron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol, CA: O'Reilly Media, Inc.

Am studiat în mod detaliat aceste cărți și am aplicat cunoștințele în acest proiect de deep learning și prelucrare a imaginilor digitale. Aceste cărți au oferit o perspectivă bogată și detaliată asupra tehnologiilor și conceptelor de deep learning și prelucrare a imaginilor digitale, oferindu-mi o bază solidă în domeniu.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

3. Analiză, proiectare, implementare

Distributia si clasificarea datelor:

În multe domenii de activitate, analiza datelor este esențială pentru a descoperi modele și tendințe care ar putea fi utilizate pentru a îmbunătăți produse sau servicii. Pentru a putea efectua analize complexe și învățare automată, datele trebuie să fie structurate și organizate într-un mod coerent și accesibil. În acest sens, unul dintre cele mai utilizate moduri de organizare a datelor este prin împărțirea acestora în două foldere mari: training și testing.

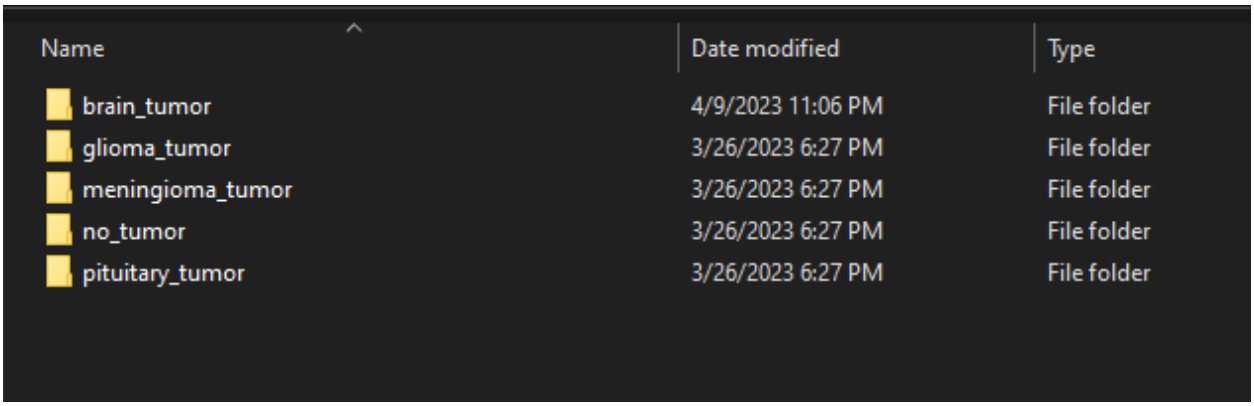
Folderele training și testing sunt utilizate pentru a ajuta la instruirea și testarea algoritmilor de învățare automată. Folderul de training conține datele pe care algoritmul de învățare automată le utilizează pentru a învăța modele, iar folderul de testing conține datele pe care algoritmul le utilizează pentru a testa eficiența modelului creat.

Structurarea datelor în foldere separate pentru training și testing este esențială pentru a asigura că algoritmul de învățare automată nu se bazează doar pe datele de training și poate generaliza și aplica modelele dobândite în mod eficient la date noi. Acesta este un aspect critic în crearea unui model care poate fi utilizat într-un mediu de producție real.

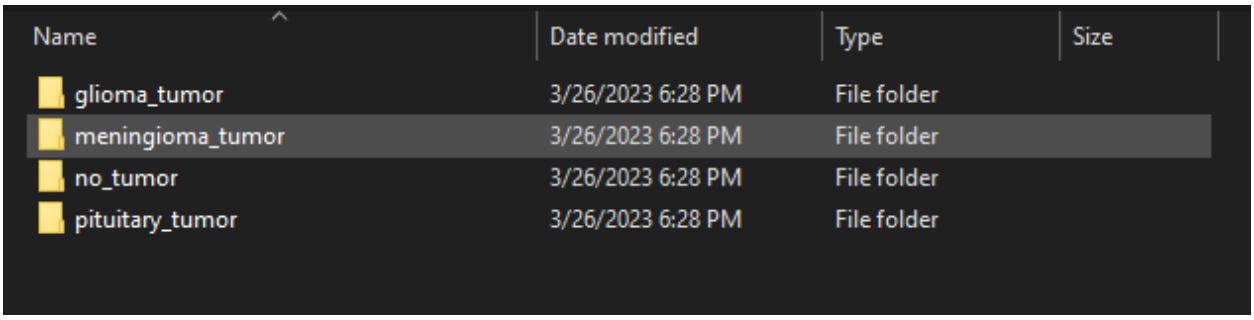
În general, folderul de training conține o cantitate mare de date care sunt utilizate pentru a instrui algoritmul de învățare automată. După ce algoritmul este antrenat, folderul de testing este utilizat pentru a evalua performanța acestuia pe date noi. În general, se recomandă ca folderul de testing să conțină cel puțin 20% din datele disponibile pentru a asigura că modelul are capacitatea de a generaliza.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Structura datelor de testare



Structura datelor de training



Avem urmatoarele date pentru training:

glioma_tumor ~= 826

meningioma_tumor ~= 822

no_tumor ~= 395

pituitary_tumor ~= 827

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Metode de preprocesare a datelor

1) Colectarea datelor

```
def collect_images_from_files(X, Y, resize_value = STANDARD_RESIZE_VALUE):
    classifier_classes = {'no_tumor': 0, 'glioma_tumor': 1, 'pituitary_tumor': 1,
'meningioma_tumor': 1}

    for cls_class in classifier_classes:
        path = STANDARD_DATA_DIRECTORY_STRUCTURE + cls_class
        for file in os.listdir(path):
            img = cv2.imread(path + '/' + file, 0)
            img = cv2.resize(img, (resize_value, resize_value))
            X.append(img)
            Y.append(classifier_classes[cls_class])
```

- Colectarea imaginilor din folderul unde sunt stocate. Pentru fiecare poza se face resize la o anumita dimensiunea (Standard e setata la 300X300 px);

- Pozele sunt colectate in doua liste si anume X si Y;

Mod de utilizare:

```
X = []
Y = []

collect_images_from_files(X, Y)
```

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

2) Convertirea datelor intr-un datastracture de tip numpy.array ca sa putem face preprocesarea.

```
X = np.array(X)
Y = np.array(Y)
X.shape
```

```
(394, 300, 300)
```

3) Dupa care trebuie sa facem reshape la date, din 3D in 2D deoarece stim ca sklearn nu accepta doar date bidimensionale

```
# Sklearn accepts only bidiomensional data, so we have to convert it

X_reshape = X.reshape(len(X), -1)
X_reshape.shape
```

```
(394, 90000)
```

4) Separarea datelor in testing si training; 20% din date o sa fie pentru testare

```
xTrain, xTest, yTrain, yTest = train_test_split(X_reshape, Y, random_state=10, test_size=.2)
```

```
xTrain.shape, xTest.shape
```

```
((315, 90000), (79, 90000))
```

5) Scalam datele

```
def scale_data_tests(xTrain, xTest):
    xTrain = xTrain / 255
    xTest = xTest / 255

    return xTrain, xTest
```

```
xTrain, xTest = scale_data_tests(xTrain, xTest)
```

In urmatorul pas, datele pot sa fie transferate catre algoritmul de machine learning;

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Logistic Regression

Logistic Regression este un model matematic folosit in statistica si invatarea automata (machine learning) pentru a prezice valoarea unei variabile binare in functie de mai multe variabile de intrare (predictori). Mai exact, logistic regression se refera la regresia logistica binara, unde variabila tinta poate avea doar doua valori posibile, de exemplu "1" sau "0", "adevarat" sau "fals".

In mod obisnuit, logistic regression este utilizat pentru a face predictii in cazurile in care variabila dependenta este de tip binar, cum ar fi prezicerea daca un pacient va dezvolta sau nu o anumita boala, daca un client va cumpara sau nu un anumit produs sau daca un email este spam sau nu. Modelul calculeaza o probabilitate a evenimentului binar in functie de variabilele de intrare si apoi transforma aceasta probabilitate intr-o valoare de 0 sau 1.

In mod concret, modelul logistic regression calculeaza o functie sigmoidala a sumei ponderate a valorilor predictori, iar aceasta functie sigmoidala reprezinta probabilitatea de a avea evenimentul binar. In cazul unei probleme de clasificare binara, se alege o valoare prag, cum ar fi 0.5, pentru a separa cele doua clase.

Functia de activare Sigmoida: (Sigmoid activation)

Math

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

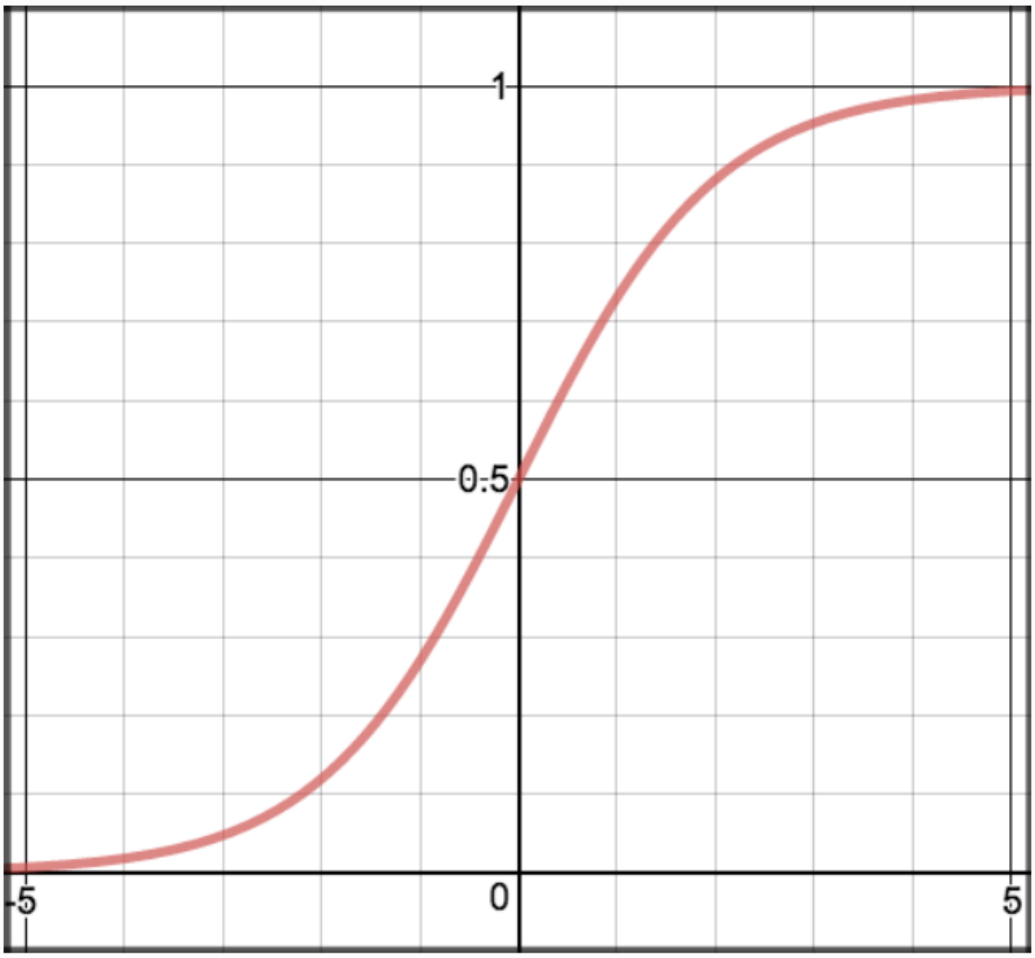
Note

- $s(z)$ = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithm's prediction e.g. mx + b)
- e = base of natural log

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Grafic si posibila implementare in Python

Graph



Code

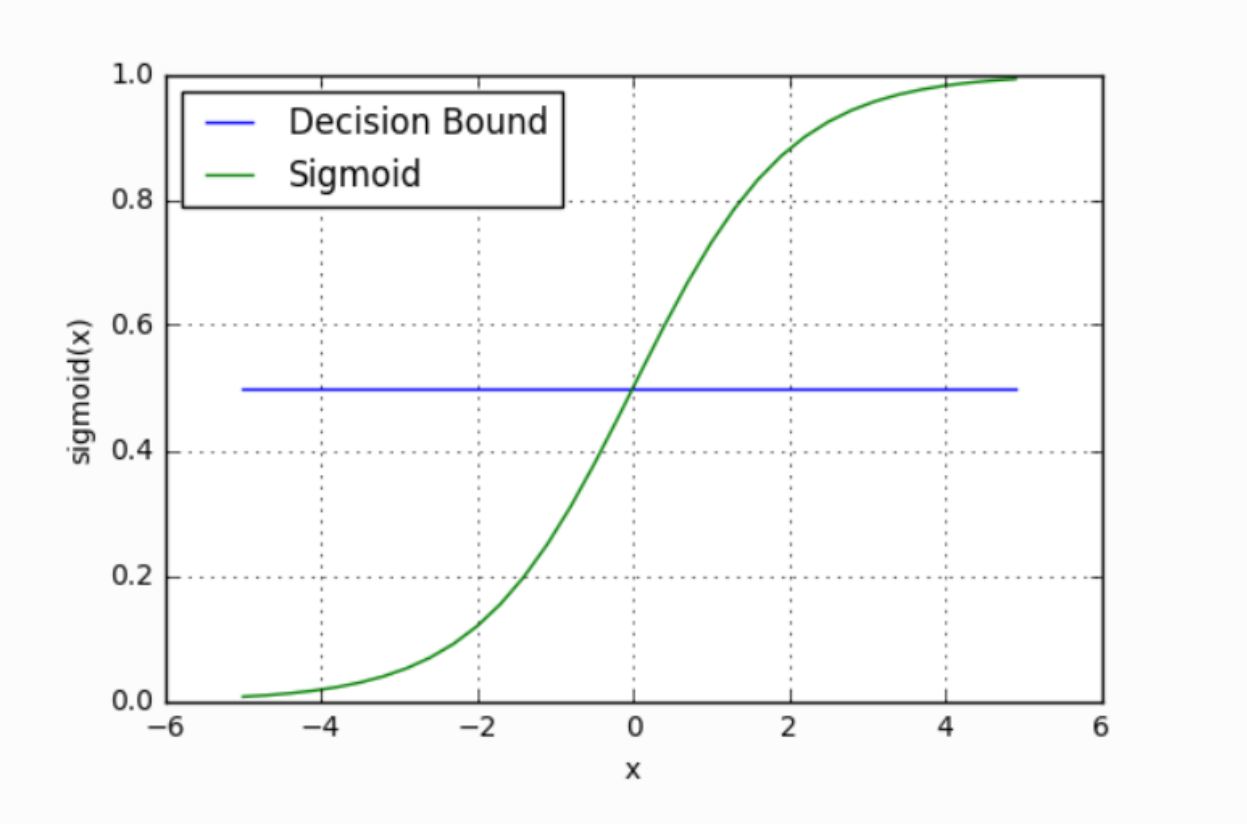
```
def sigmoid(z):
  return 1.0 / (1 + np.exp(-z))
```

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Decision Boundary

$$p \geq 0.5, class = 1$$
$$p < 0.5, class = 0$$

Un posibil Graf



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Cost function - Functia de cost este o functie matematica folosita in algoritmi de machine learning pentru a evalua cat de bine se potrivesc modelele la datele de antrenament. Scopul principal al functiei de cost este sa determine parametrii modelului astfel incat sa minimizeze eroarea dintre predictiile modelului si valorile reale.

Matematic:

$$MSE = \frac{1}{2N}\sum_{i=1}^{n}(y_i - (W_1x_1 + W_2x_2 + W_3x_3))^2$$

In esenta, functia de cost calculeaza distanta intre predictiile modelului si valorile reale din setul de date de antrenament. In functie de algoritmul de machine learning folosit, functia de cost poate fi diferita.

Gradient descent

Gradient descent este o metoda de optimizare a unei functii de cost prin iteratii succesive pentru a ajunge la valoarea minima a functiei respective. Ideea din spatele gradient descent este de a urmari gradientul functiei de cost pentru a gasi directia cea mai rapida de coborare (in cazul minimizarii functiei de cost) si apoi de a face un pas in directia respectiva. Acest proces se repeta pana cand functia de cost ajunge la o valoare minima sau cand un anumit criteriu de oprire este atins.

$$f'(W_1) = -x_1(y - (W_1x_1 + W_2x_2 + W_3x_3))$$
$$f'(W_2) = -x_2(y - (W_1x_1 + W_2x_2 + W_3x_3))$$
$$f'(W_3) = -x_3(y - (W_1x_1 + W_2x_2 + W_3x_3))$$

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Modul de implementare in proiect si parametrii de functionare:

```
[8]: from sklearn.linear_model import LogisticRegression

[9]: classifier = LogisticRegression(C=0.1)
     classifier.fit(xTrain, yTrain)
```

```
C:\Python310\lib\site-packages\sklearn\linear_model\_lo
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale t
    https://scikit-learn.org/stable/modules/preprocessi
Please also refer to the documentation for alternative
    https://scikit-learn.org/stable/modules/linear_mode
  n_iter_i = _check_optimize_result(
```

```
[9]: LogisticRegression
     LogisticRegression(C=0.1)
```

```
[10]: classifier.score(xTest, yTest)

[10]: 0.8481012658227848
```

Parametrul C reprezintă un parametru de regularizare. Regularizarea este o tehnică utilizată pentru a controla overfitting-ul în modelele de machine learning.

Parametrul C inversează forța de regularizare, ceea ce înseamnă că o valoare mică a lui C va duce la o regularizare puternică, în timp ce o valoare mare a lui C va duce la o regularizare mai slabă. Prin urmare, o valoare mai mică a lui C poate duce la un model mai simplu, care poate avea o performanță mai bună pe datele de testare, în timp ce o valoare mai mare a lui C poate duce la un model mai complex, care poate avea o performanță mai bună pe datele de antrenament.

Cativa parametri default folositi de LogisticRegression si care merita mentionati:

max_iter default ~= **100**

multi_class default ~= **auto**

solver default ~= **Limited-memory BFGS**

Lista cu parametrii care ii poate lua LogisticRegression:

- penalty (default='l2'): specifică tipul de penalizare folosit pentru a controla overfitting-ul. Poate fi 'l1', 'l2', 'elasticnet', sau None.
- dual (default=False): specifică dacă să folosească formularea primală sau duală a problemelor de optimizare.
- tol (default=1e-4): specifică toleranța pentru criteriul de oprire.
- C (default=1.0): specifică puterea inversă a regularizării. A fost discutat mai sus.
- fit_intercept (default=True): specifică dacă să se potrivească cu un intercept (bias) pentru regresie.
- intercept_scaling (default=1): specifică scara interceptului, dacă fit_intercept este setat la True.
- class_weight (default=None): specifică greutatea asociată fiecărei clase. Poate fi None sau 'balanced'.
- random_state (default=None): specifică seed-ul generatorului de numere aleatoare.
- solver (default='lbfgs'): specifică algoritmul folosit pentru optimizarea problemei. Poate fi 'newton-cg', 'lbfgs', 'liblinear', 'sag', sau 'saga'.
- max_iter (default=100): specifică numărul maxim de iterații pentru solverul ales.
- multi_class (default='auto'): specifică strategia de gestionare a problemelor multi-clasa. Poate fi 'ovr' (one-vs-rest) sau 'multinomial'.
- verbose (default=0): specifică nivelul de verbositate al solverului.
- warm_start (default=False): specifică dacă să se folosească soluția anterioară ca punct de plecare.
- n_jobs (default=None): specifică numărul de nuclee procesor disponibile pentru a rula în paralel. Poate fi None sau un intreg.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Random Forest Classifier

Ideea de bază a algoritmului Random Forest este de a crea un set de arbori de decizie, unde fiecare arbore de decizie este construit prin eșantionarea aleatorie a datelor și a variabilelor (numite și caracteristici sau "features"). Acest proces de eșantionare aleatorie este cunoscut sub numele de bagging (bootstrap aggregating).

Prin bagging, algoritmul va crea mai multe eșantioane aleatorii din setul de date inițial. Fiecare eșantion va avea același număr de exemple din setul de date inițial, dar fiecare exemplu va fi ales aleatoriu, cu înlocuire, astfel încât aceleași exemple pot apărea de mai multe ori în același eșantion. Apoi, un arbore de decizie este construit pe fiecare eșantion și aceste arbori de decizie formează ansamblul final de arbori.

Atunci când se face o predicție, fiecare arbore de decizie din ansamblu clasifică punctul de date (sau observația) și votul majoritar al acestor arbori va fi ales ca etichetă de clasă a punctului de date.

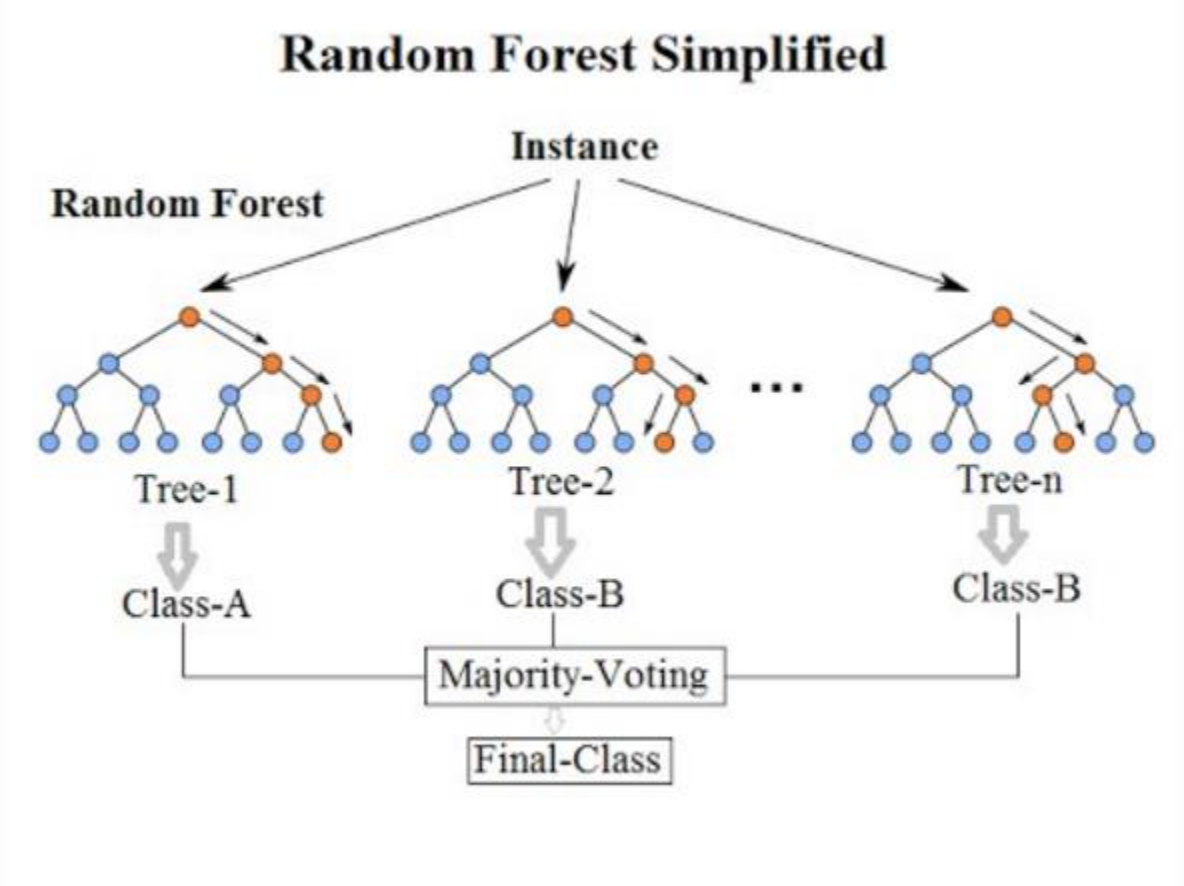
Această abordare are mai multe avantaje, printre care:

Reducerea overfitting-ului, adică obținerea unor modele mai generalizate, capabile să se descurce bine cu noi date de testare care nu au fost incluse în setul de antrenament.

Capacitatea de a face față seturilor de date mari, deoarece eșantionarea aleatorie reduce timpul de antrenare și de predicție și poate îmbunătăți acuratețea.

Capacitatea de a gestiona atât caracteristici continue, cât și discontinue, fără a necesita preprocesarea sau normalizarea acestora.

Mai jos este o diagrama simplificata care ilustreaza cum functioneaza acest algoritm de supervised learning.



Bagging - bootstrap aggregating

Procedura de bagging constă în eșantionarea repetată cu înlocuire a setului de date de antrenament inițial și apoi construirea mai multor modele pe baza acestor eșantioane. De fiecare dată când se eșantionează, se va obține un nou set de date, care va fi diferit de setul de antrenament inițial și de celelalte seturi de date eșantionate anterior.

Fiecare model construit pe baza acestor eșantioane va fi diferit de celelalte modele, deoarece setul de date utilizat pentru antrenare este diferit. Prin urmare, modelele vor

avea o variație redusă și o capacitate mai mare de a generaliza, ceea ce va duce la o mai mare acuratețe și robustețe.

După ce toate modelele sunt construite, se combină predicțiile acestora prin vot majoritar pentru problemele de clasificare sau prin medie pentru problemele de regresie, astfel încât să obținem o predicție finală mai precisă.

Formula pentru colectarea tuturor predictilor intr-o medie a lor:

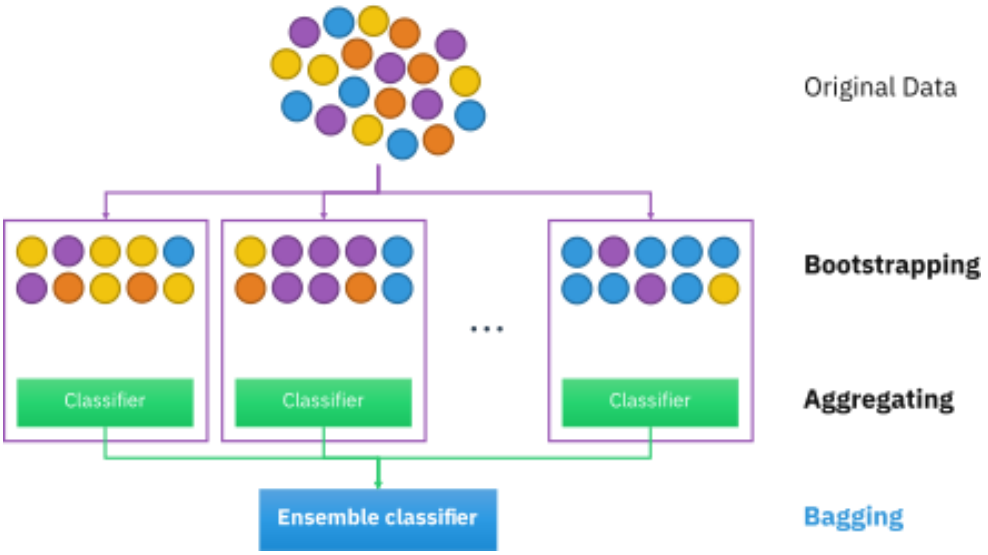
$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

În plus, se poate face o estimare a incertitudinii predicției ca deviația standard a predicțiilor de la toți arborii de regresie individuali pe x'.

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

Bagging-ul este adesea utilizat cu arbori de decizie, dar poate fi utilizat cu succes și cu alte modele precum K-NN, SVM sau rețele neurale.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	



Modelul implementat in proiect si parametrii folositi:

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(xTrain, yTrain)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=2, random_state=0)
```

```
classifier.score(xTest, yTest)
```

```
0.810126582278481
```

Parametrul max_depth este un hyperparametru care controlează adâncimea maximă a arborilor de decizie în cadrul ansamblului de arbori Random Forest. Aceasta înseamnă că se va limita numărul de nivele ale arborilor și astfel se va încerca să se prevină overfitting-ul. Un arbore cu o adâncime mare poate fi prea complex și va fi prea

specific pentru datele de antrenare, ceea ce va duce la o performanță slabă în timpul testării. In cazul meu, am ales sa fie 2 dupa repetate teste facute.

Parametrul random_state controlează reproducibilitatea modelului și face ca modelul să fie repetabil. Aceasta înseamnă că, dacă rulați același cod cu același parametru random_state, veți obține întotdeauna același model. Acest parametru este important deoarece modelul Random Forest se construiește prin eșantionarea aleatorie a setului de date și a caracteristicilor, astfel încât, fără un parametru random_state constant, modelul poate fi diferit de fiecare dată când este antrenat.

Parametrii care pot sa fie transmisi la acest model sunt o multitudine, ii mentionez mai jos:

- n_estimators: numărul de arbori de decizie din ansamblu. Cu cât este mai mare acest număr, cu atât modelul va fi mai precis, dar va avea nevoie de mai mult timp pentru a fi antrenat.
- criterion: funcția de măsurare a calității împărțirii nodurilor. Valorile acceptate sunt "gini" pentru indicele Gini și "entropy" pentru entropia informațională.
- max_depth: adâncimea maximă a arborilor de decizie. Această valoare controlează complexitatea modelelor și poate fi utilizată pentru a evita overfitting-ul. (A fost discutat si mai sus)
- min_samples_split: numărul minim de mostre necesare pentru a împărți un nod. Dacă un nod are mai puține de aceste mostre, atunci nu va mai fi posibilă împărțirea acestuia.
- min_samples_leaf: numărul minim de mostre necesare pentru a fi considerate frunze. Dacă un nod are mai puține de aceste mostre, atunci nu va mai fi posibilă împărțirea acestuia.
- min_weight_fraction_leaf: fracțiunea minimă din greutatea totală a eșantioanelor necesară pentru a fi considerate frunze.
- max_features: numărul maxim de caracteristici care trebuie luate în considerare pentru împărțirea fiecărui nod. Acest parametru poate fi util pentru a controla variația și pentru a îmbunătăți performanța generală a modelului.
- max_leaf_nodes: numărul maxim de frunze dintr-un arbore. Acest parametru poate fi utilizat pentru a controla complexitatea modelului și poate fi o alternativă la parametrul max_depth.

- min_impurity_decrease: o valoare prag pentru reducerea impurității necesară pentru a împărți un nod.
- bootstrap: dacă setat la True, atunci va fi utilizată eșantionarea bootstrap la construirea fiecărui arbore din ansamblu.
- oob_score: dacă setat la True, atunci se va calcula scorul out-of-bag pentru fiecare arbore în timpul antrenării.
- n_jobs: numărul de job-uri paralele pentru a fi utilizate în timpul antrenării. Setarea acestui parametru la -1 va utiliza toate procesele disponibile.
- random_state: un număr întreg care controlează aleatorizarea modelului. Dacă acest parametru este setat la o valoare constantă, atunci modelul va fi identic la fiecare rulare. (Din nou, a fost discutat)

SVM SVC – Support Vector Machine | Support Vector Classifier

SVM sau Support Vector Machines este un algoritm de învățare supervizată pentru clasificarea și regresia datelor. Algoritmul SVM încearcă să găsească un hiperplan în spațiul de caracteristici care să împartă datele în două clase. În cazul clasificării binare, acest hiperplan se numește plan de decizie și trebuie să fie capabil să separe datele pozitive de cele negative în spațiul de caracteristici.

SVC (Support Vector Classification) este o implementare a algoritmului SVM în biblioteca scikit-learn din Python. Acesta poate fi utilizat pentru a clasifica datele în două sau mai multe clase.

Pentru a explica modul în care funcționează SVC, vom începe prin a defini un concept important al algoritmului SVM: marginile. Marginile sunt spațiile libere lăsate între hiperplanul de decizie și punctele cele mai apropiate de fiecare parte a acestuia. Scopul algoritmului SVM este de a maximiza aceste margini.

În cazul clasificării binare, SVC încearcă să găsească hiperplanul de decizie care maximizează aceste margini. Aceasta se realizează prin găsirea unui set de vectori suport, care sunt punctele cele mai apropiate de hiperplanul de decizie. Acești vectori suport sunt folosiți pentru a defini hiperplanul de decizie, iar distanța dintre hiperplan și vectorii suport reprezintă marginile.

Algoritmul SVC utilizează o funcție de kernel pentru a transforma datele din spațiul de caracteristici original într-un alt spațiu de caracteristici, unde poate fi mai ușor să se găsească un hiperplan de decizie. Cele mai utilizate funcții de kernel sunt:

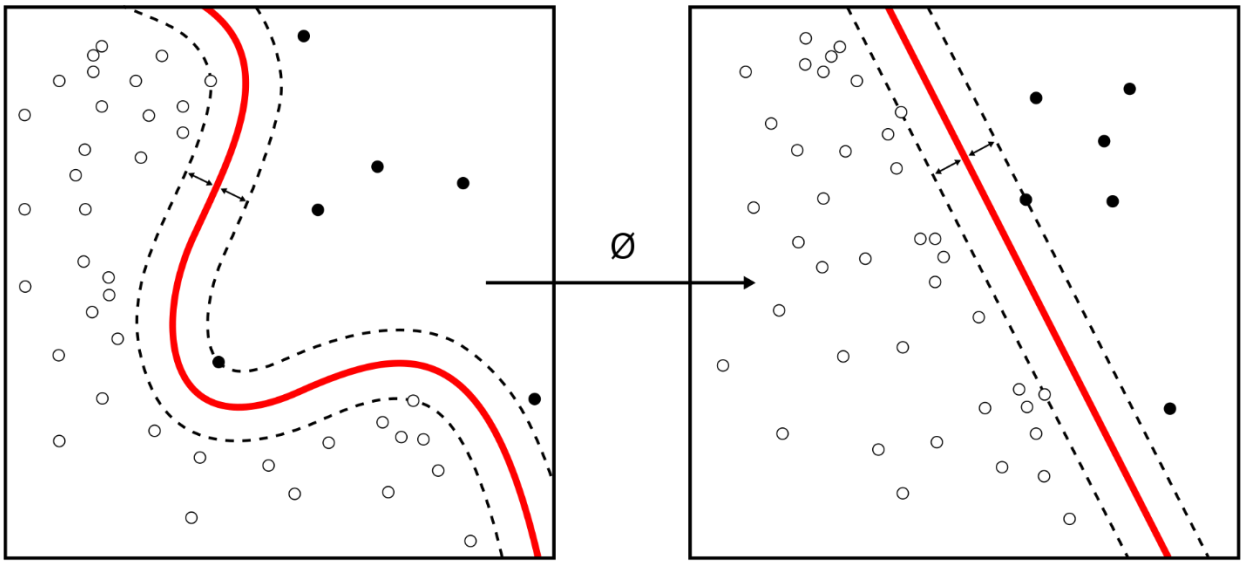
Linear: această funcție de kernel calculează produsul scalar dintre două vectori. Este folosită pentru seturi de date liniar separabile.

Polynomial: această funcție de kernel calculează produsul scalar ridicat la puterea unui anumit grad. Este utilizată pentru seturi de date non-liniar separabile.

RBF (Radial Basis Function): această funcție de kernel transformă datele într-un spațiu de caracteristici infinite-dimensional bazat pe distanța euclidiană dintre puncte. Este foarte utilă pentru seturi de date non-liniar separabile.

În general, algoritmul SVC poate fi utilizat pentru a clasifica datele în două sau mai multe clase. În cazul clasificării multiclase, SVC utilizează o strategie one-vs-one sau one-vs-rest pentru a clasifica datele. În strategia one-vs-one, SVC construiește un clasificator binar pentru fiecare pereche de clase. În strategia one-vs-rest, SVC construiește un clasificator binar pentru fiecare clasă în parte, tratând celelalte clase ca fiind negative.

Aplicatii ale Kernelului

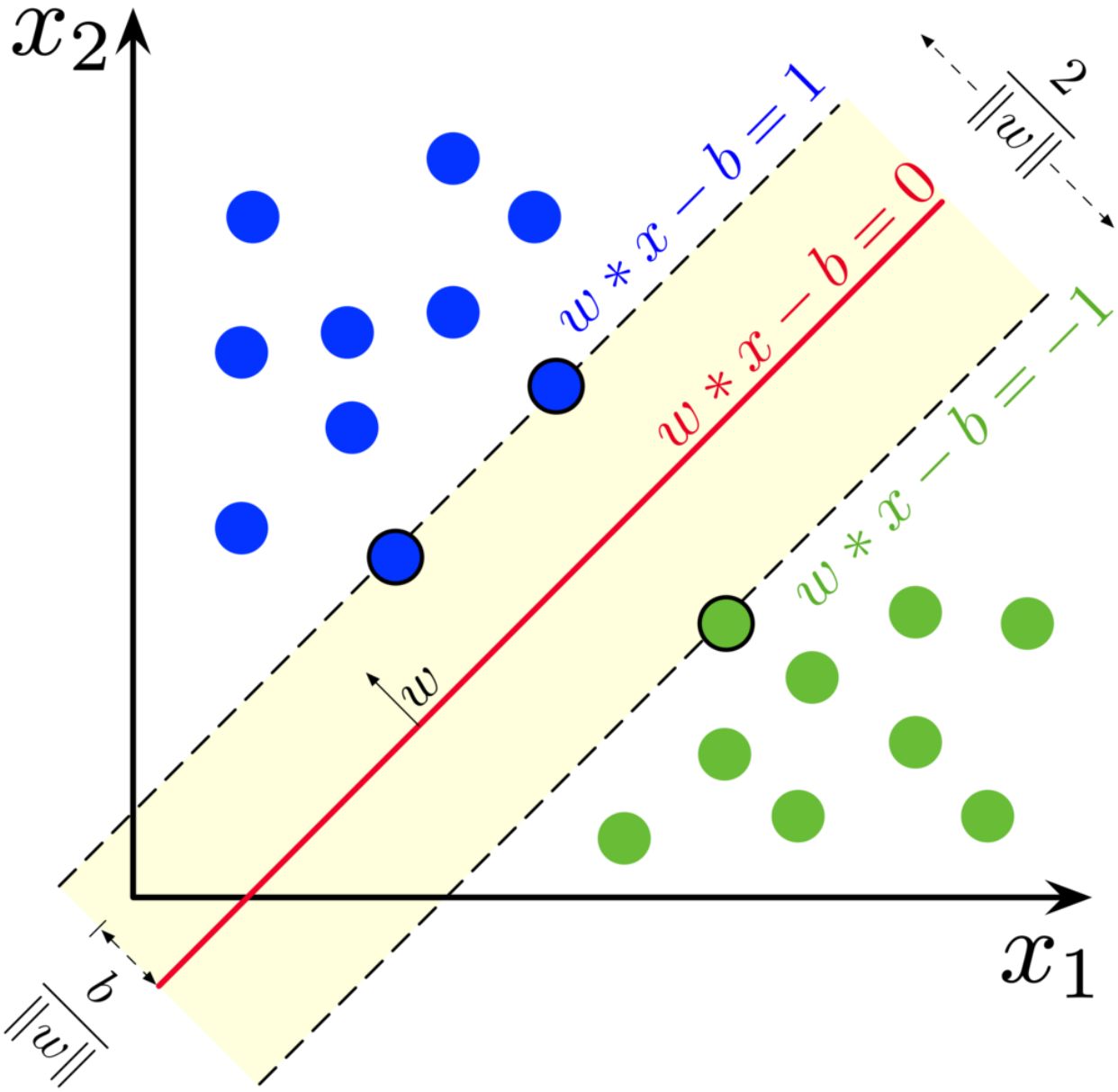


PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Kernel Linear:

Kernelul linear este unul dintre cei mai simpli kerneli folosiți în suportul vectorial de clasificare (SVC). Acesta este utilizat pentru a clasifica datele liniar separabile, adică date care pot fi separate printr-o linie dreaptă într-un spațiu multidimensional.

Ideea din spatele kernelului linear este de a proiecta datele de intrare într-un nou spațiu dimensional, unde acestea pot fi separate printr-o linie dreaptă. Aceasta se face prin multiplicarea fiecărei caracteristici de intrare cu un anumit coeficient și apoi adunarea acestora pentru a obține un punct în noul spațiu dimensional. Dacă datele de intrare au m caracteristici, atunci acest proces de proiecție va rezulta într-un nou set de date cu m dimensiuni.



Soft-margin si Hard-Margin:

Hard-margin SVM încearcă să găsească un hiperplan care separă perfect cele două clase de date, adică nu permite nici o eroare de clasificare. Acest lucru

funcționează bine atunci când datele sunt complet separabile linear. Însă, în practică, datele nu sunt întotdeauna complet separabile și, prin urmare, nu se poate găsi un hiperplan perfect de separare.

Concluzie matematica:

$$\begin{aligned} \underset{\mathbf{w},\, b}{\text{minimize}} \quad & \|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Soft-margin SVM, pe de altă parte, permite unele erori de clasificare, dar încearcă totuși să găsească un hiperplan de separare cu o marjă maximă. Marja este distanța dintre hiperplan și cel mai apropiat punct de date din cele două clase. Soft-margin SVM permite unele puncte să fie situate în interiorul acestei marje, dar încearcă totuși să maximizeze dimensiunea acestei marje.

Concluzie matematica:

$$\begin{aligned} \underset{\mathbf{w},\, b,\, \zeta}{\text{minimize}} \quad & \|\mathbf{w}\|_2^2 + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Kernel non-linear:

Kernelul non-liniar este o tehnică utilizată în (SVM) pentru a clasifica datele care nu sunt liniar separabile, adică date care nu pot fi separate printr-un hiperplan liniar. În loc să încerce să găsească un hiperplan liniar de separare, kernelul non-liniar proiectează datele de intrare într-un spațiu dimensional mai mare în care acestea pot fi separabile printr-un hiperplan liniar.

Proiecția datelor într-un spațiu dimensional mai mare se face prin utilizarea unei funcții de proiecție non-liniară. Această funcție poate fi o funcție polinomială sau o funcție radială, de exemplu. Funcția de proiecție transformă datele de intrare într-un spațiu dimensional mai mare în care acestea sunt liniar separabile.

Există mai multe tipuri de kerneluri non-liniare folosite în (SVM) pentru a clasifica datele care nu sunt liniar separabile. Iată câteva exemple de kerneluri non-liniare utilizate în mod obișnuit:

Kernelul polinomial: Acesta este un tip de kernel non-liniar care utilizează o funcție polinomială pentru a proiecta datele de intrare într-un spațiu dimensional mai mare. Funcția de proiecție este de obicei o funcție de gradul doi sau trei. Kernelul polinomial este utilizat în mod obișnuit în problemele de clasificare cu două sau mai multe clase.

Kernelul radial: Acesta este un tip de kernel non-liniar care utilizează o funcție radială pentru a proiecta datele de intrare într-un spațiu dimensional mai mare. Funcția de proiecție se bazează pe distanța dintre datele de intrare și un punct central. Kernelul radial este utilizat în mod obișnuit în problemele de clasificare binară.

Kernelul sigmoidal: Acesta este un tip de kernel non-liniar care utilizează o funcție sigmoidală pentru a proiecta datele de intrare într-un spațiu dimensional mai mare. Funcția de proiecție se bazează pe funcția sigmoidală, care transformă datele de intrare într-o formă care poate fi separată liniar. Kernelul sigmoidal este utilizat în mod obișnuit în problemele de clasificare binară.

Acestea sunt doar câteva exemple de kerneluri non-liniare utilizate în SVM. Există și alți kerneluri non-liniari, cum ar fi kernelul exponential și kernelul Laplace, care sunt utilizați în funcție de natura specifică a datelor și a problemei de clasificare.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Considerent matematic:

Polinomial omogen:

$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$ Cand d este 1 atunci sistemul devine linear

Polinomial neomogen:

$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + r)^d.$

Radial Basis Function (Ceea ce folosim si noi):

$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$ pentru $\gamma > 0.$

Sigmoid – Tangenta hiperbolica:

$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x}_i \cdot \mathbf{x}_j + c)$

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Modelul implementat in proiect si parametrii folositi:

```
from sklearn.svm import SVC
```

```
classifier = SVC(C=100)
classifier.fit(xTrain, yTrain)
```

```
SVC
SVC(C=100)
```

```
classifier.score(xTest, yTest)
```

```
0.9367088607594937
```

C: Parametrul de regularizare C controlează costul unei erori de clasificare. Valori mai mari ale lui C duc la modele mai rigide, care pot duce la o acuratețe mai mare a antrenării, dar pot fi sensibile la overfitting. Valori mai mici ale lui C conduc la modele mai puțin rigide, care sunt mai flexibile și mai puțin sensibile la overfitting. Valoarea implicită este 1.0

kernel: Kernelul specifică funcția de transformare a datelor de intrare într-un spațiu cu dimensiuni mai mari, unde acestea pot fi mai ușor clasificate. Exemple de kerneluri includ kernelul liniar, kernelul polinomial și kernelul radial (RBF). Valoarea implicită este kernelul RBF.

degree: Gradul polinomului pentru kernelul polinomial. Acest parametru este utilizat numai dacă kernelul este setat la "poly". Valoarea implicită este 3.

gamma: Parametrul gamma controlează cât de departe ajunge influența unui singur punct de date. Valori mari ale lui gamma conduc la modele cu o acuratețe mai mare a antrenării, dar pot fi sensibile la overfitting. Valori mai mici ale lui gamma conduc la

modele mai puțin rigide, care sunt mai flexibile și mai puțin sensibile la overfitting. Valoarea implicită este "scale".

coef0: Termenul independent din funcția kernel. Acest parametru este utilizat numai dacă kernelul este setat la "poly" sau "sigmoid". Valoarea implicită este 0.0

shrinking: Parametrul shrinking controlează dacă algoritmul va utiliza tehnicile de "shrinking" pentru a accelera viteza de antrenare. Valoarea implicită este True

probability: Parametrul probability controlează dacă algoritmul va produce probabilități de clasificare în loc de valori binare. Valoarea implicită este False.

tol: Toleranța pentru criteriul de oprire. Antrenarea se oprește atunci când modificarea valorii funcției obiectiv este mai mică decât această toleranță. Valoarea implicită este 1e-3.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Metoda de testare a datelor si afisare:

```
def test_based_on_images(classifier, fileTestingClass, size = 9, resize_value =
STANDARD_RESIZE_VALUE):
    values_classifier = {0: 'No Tumor', 1: 'Is Tumor'}

    plt.figure(figsize=(12, 8))
    path = os.listdir(STANDARD_DATA_DIRECTORY_STRUCTURE)

    subplot_index = 1

    for i in os.listdir(STANDARD_DATA_DIRECTORY_STRUCTURE + fileTestingClass)[:size]:
        plt.subplot(3, 3, subplot_index)

        img = cv2.imread(STANDARD_DATA_DIRECTORY_STRUCTURE + fileTestingClass + i, 0)
        img_resized = cv2.resize(img, (resize_value, resize_value))
        img_resized = img_resized.reshape(1, -1) / 255
        path = classifier.predict(img_resized)
        plt.title(values_classifier[path[0]])
        plt.imshow(img, cmap = 'gray')
        plt.axis('off')

        if subplot_index == 9:
            subplot_index = 1
        else:
            subplot_index += 1
```

- Clasificarea e binara, fie exista tumoare fie nu este;
- Se ia folderul de testing, care sunt date separate si se face o testare pe acestea;
- Modul de display este 3X3, si se poate seta care sa fie starting-point-ul de unde sa se faca testarea dintre toate datele de testing;

4. Concluzii

a. Rezultate obţinute

In cadrul proiectului de detectare a tumorilor cerebrale, s-au utilizat trei algoritmi de machine learning: LogisticRegression, SVM si RandomForestClassifier. Pentru a evalua performanta acestor algoritmi, s-a folosit o imagine standard de marime 300x300.

Dupa analiza datelor, s-a observat ca LogisticRegression a avut o performanta de 0.8481012658227848, ceea ce reprezinta o acuratete de 85%. RandomForestClassifier a avut o performanta de 0.810126582278481, ceea ce reprezinta o acuratete de 81%. In schimb, SVM a obtinut o performanta impresionanta de 0.9367088607594937, adica aproape 94% acuratete.

Aceste rezultate arata ca SVM a fost superior in analiza imaginilor si in detectarea tumorilor cerebrale, o performanta remarcabila poate fi atribuita capacitatii acestui algoritm de a gestiona date complexe si de a detecta modele mai precise. In consecinta, SVM este considerat a fi algoritmul ideal pentru a fi utilizat in cadrul sistemului de detectare a tumorilor cerebrale, deoarece ofera o acuratete buna si o performanta superioara in comparatie cu ceilalti doi algoritmi testati.

b. Direcţii de dezvoltare

Sistemul dezvoltat in cadrul acestui proiect pentru detectarea anomaliilor cerebrale reprezinta o solutie promitatoare pentru diagnosticarea si tratamentul acestora. Cu toate acestea, exista inca oportunitati de cercetare care pot imbunatati performanta sistemului si pot face posibila detectarea mai precisa a anomaliilor.

Una dintre aceste directii de cercetare este antrenarea sistemului cu date noi si mai complexe. Radiografiile din diferite unghiuri si perspective pot fi incluse in dataset-ul de antrenament pentru a imbunatati performanta sistemului in detectarea anomaliilor.

Aceasta ar putea creste acuratetea sistemului si ar face posibila detectarea anomaliilor mai mici si mai putin vizibile.

De asemenea, exista oportunitati de cercetare in procesul de preprocesare a imaginilor. Sistemul poate fi imbunatatit prin identificarea si utilizarea unor metode mai precise de preprocesare a imaginilor, astfel incat sa se asigure ca input-ul oferit algoritmului de machine learning este cel mai potrivit posibil. Aceste cercetari ar putea include identificarea celor mai bune tehnici de preprocesare pentru reducerea zgomotului sau a distorsiunilor, eliminarea artefactelor si imbunatatirea calitatii imaginii.

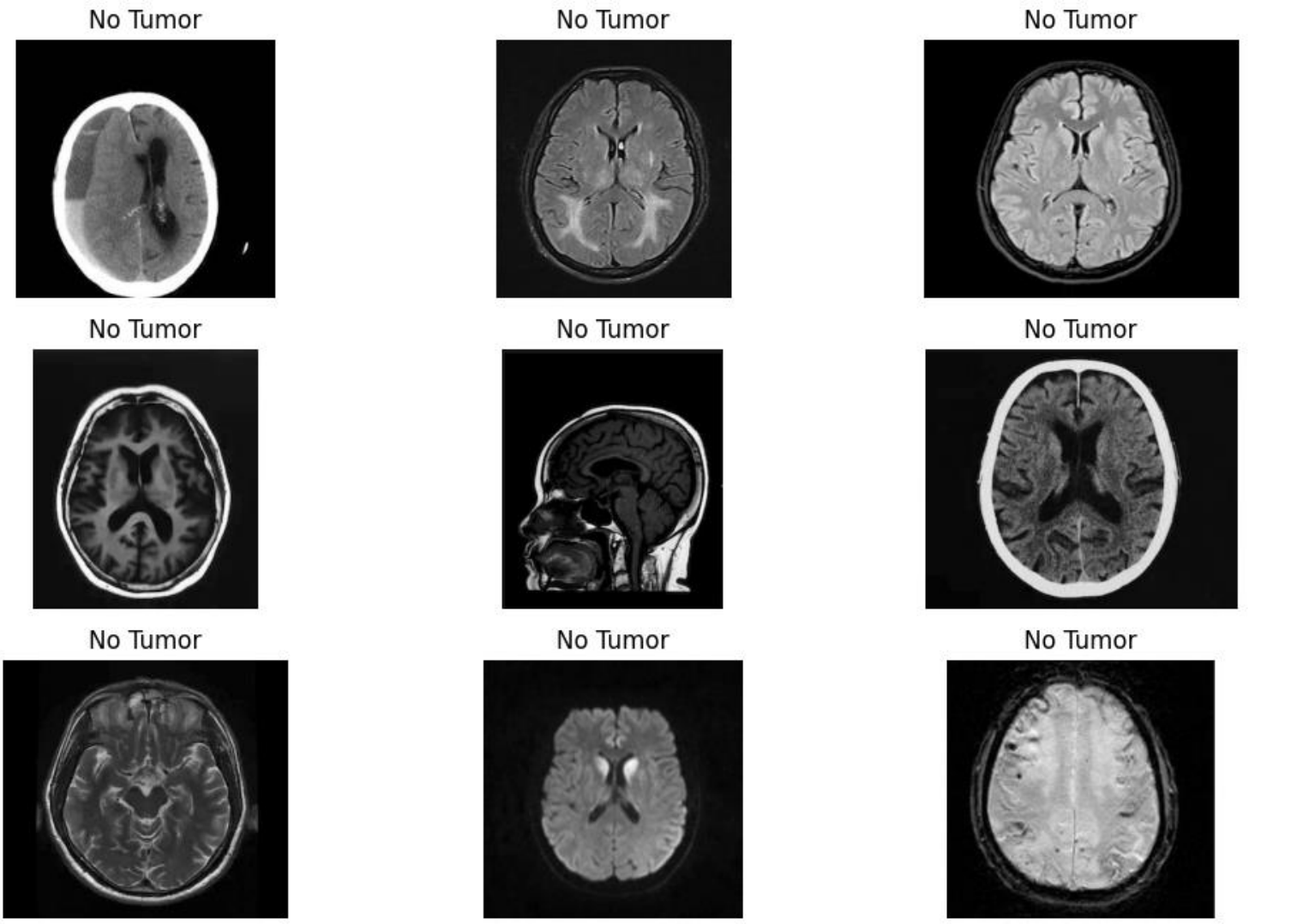
In concluzie, directiile viitoare de cercetare pentru sistemul de detectare a anomaliilor cerebrale includ imbunatatirea dataset-ului de antrenament si procesul de preprocesare a imaginilor. Aceste cercetari ar putea imbunatati semnificativ performanta sistemului si ar face posibila detectarea mai precisa a anomaliilor. Prin urmare, acest proiect reprezinta doar un prim pas in dezvoltarea unor solutii mai precise si mai eficiente pentru diagnosticarea si tratamentul anomaliilor cerebrale.

PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

APPENDIX A.

No Tumor - Logistic Regression:

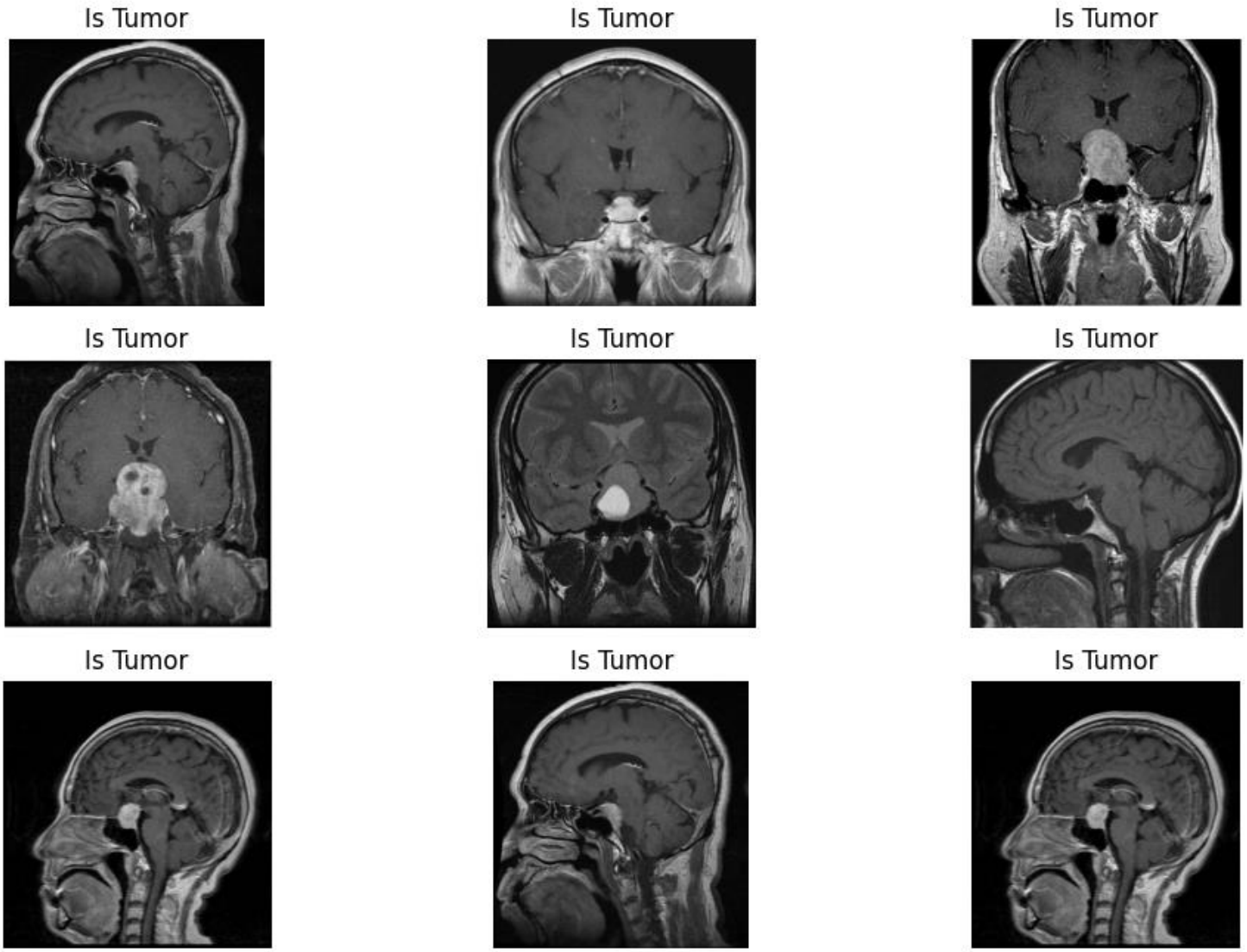
```
[11]: test_based_on_images(classifier, 'no_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Pituitary Tumor Detection - Logistic Regression:

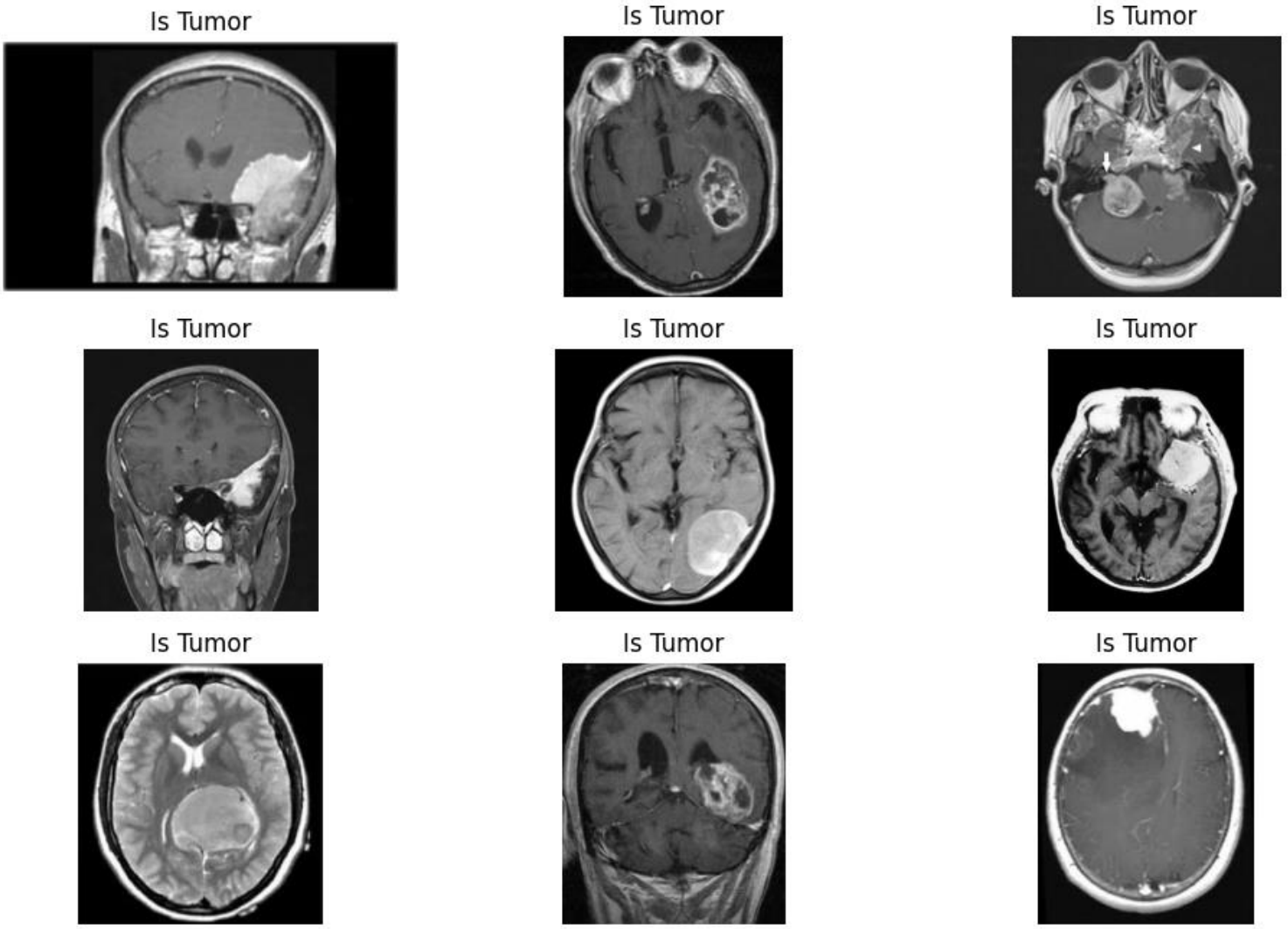
```
[12]: test_based_on_images(classifier, 'pituitary_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

MeningiomaTumor Detection – Logistic Regression

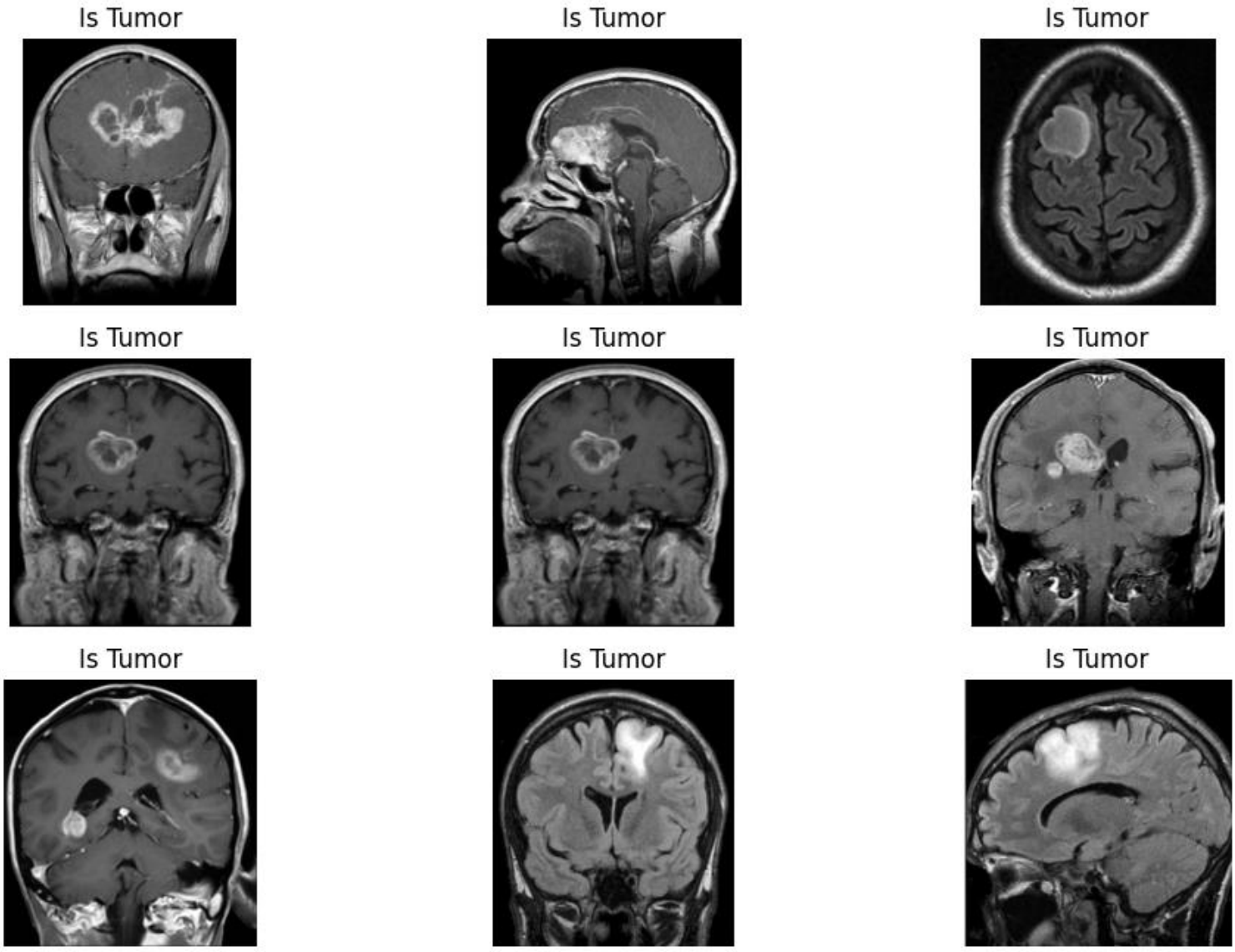
```
[13]: test_based_on_images(classifier, 'meningioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Glioma Tumor Detection - Logistic Regression

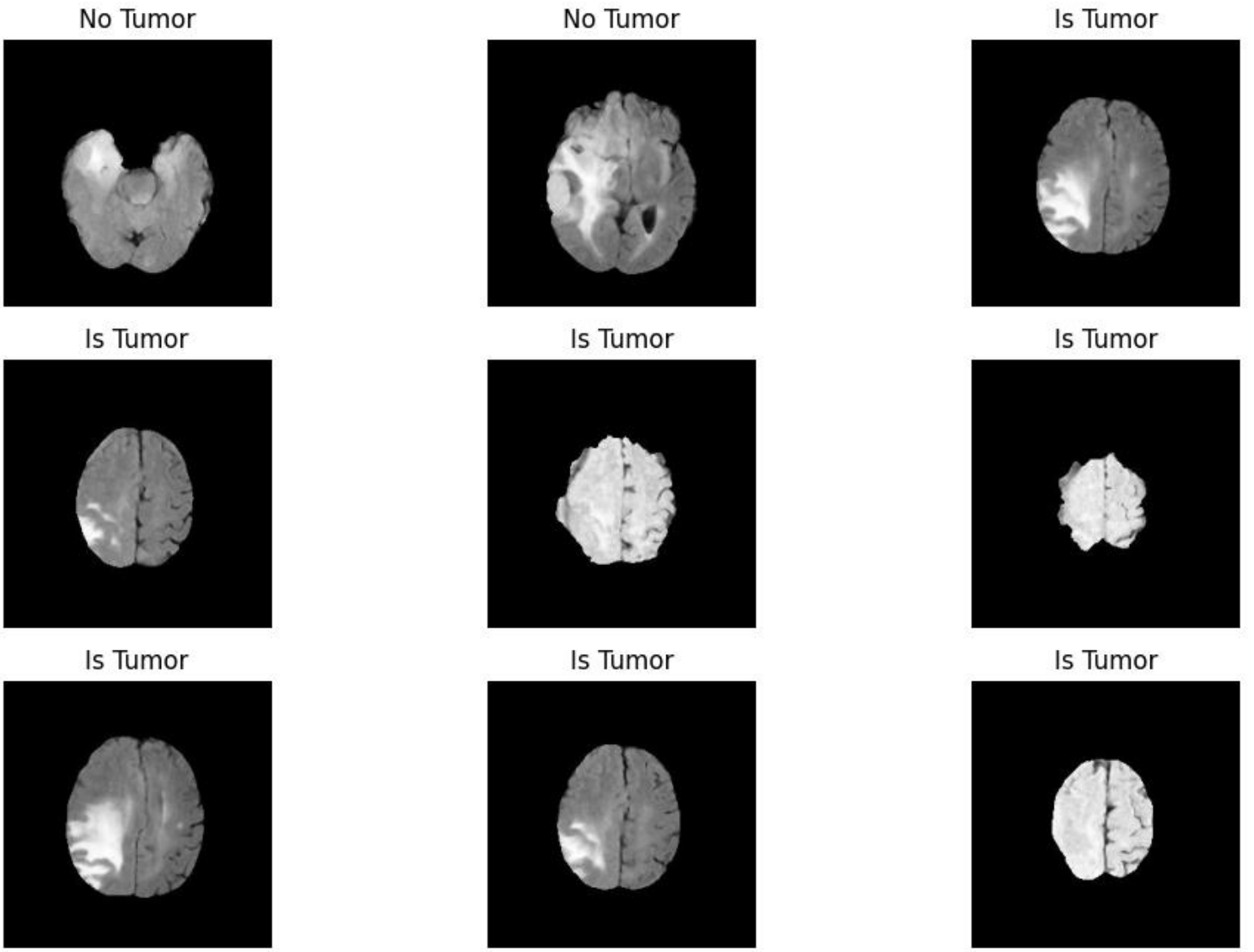
```
[14]: test_based_on_images(classifier, 'glioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

New Tumor Dataset Detection - Logistic Regression

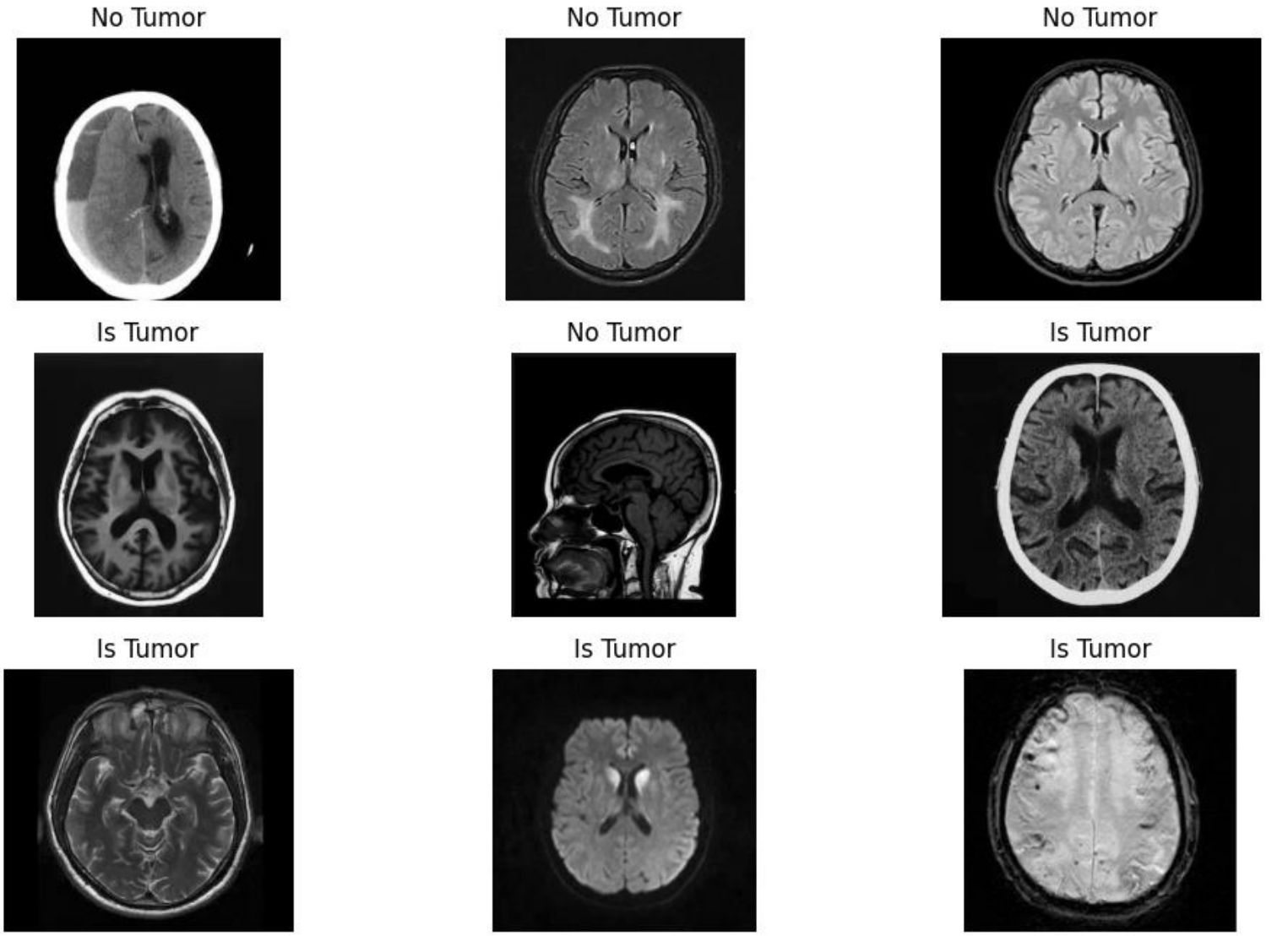
[15]:
```
test_based_on_images(classifier, 'brain_tumor/', 400)
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

No Tumor - Random Forest Classifier:

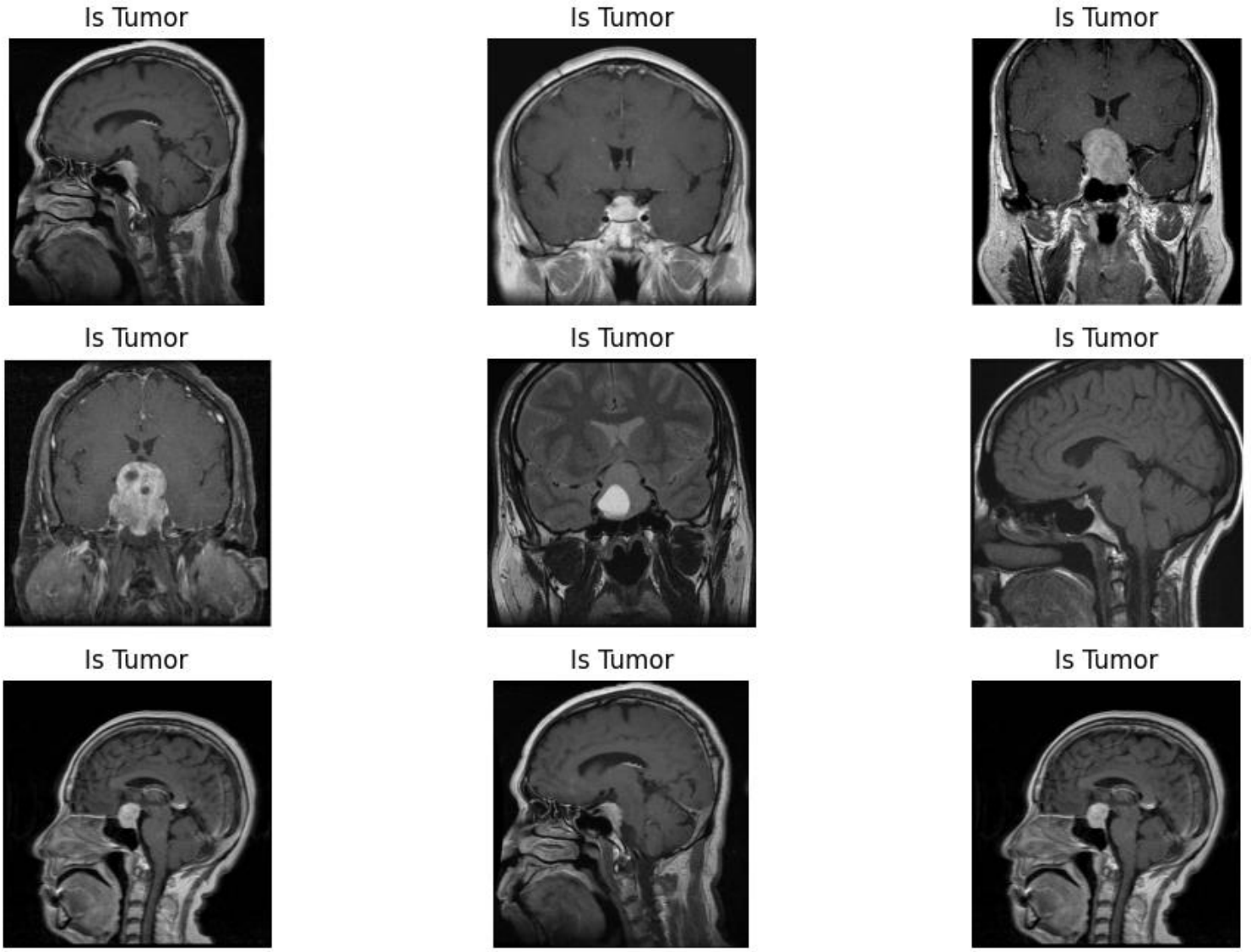
```
[11]: test_based_on_images(classifier, 'no_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Pituitary Tumor Detection - Random Forest Classifier:

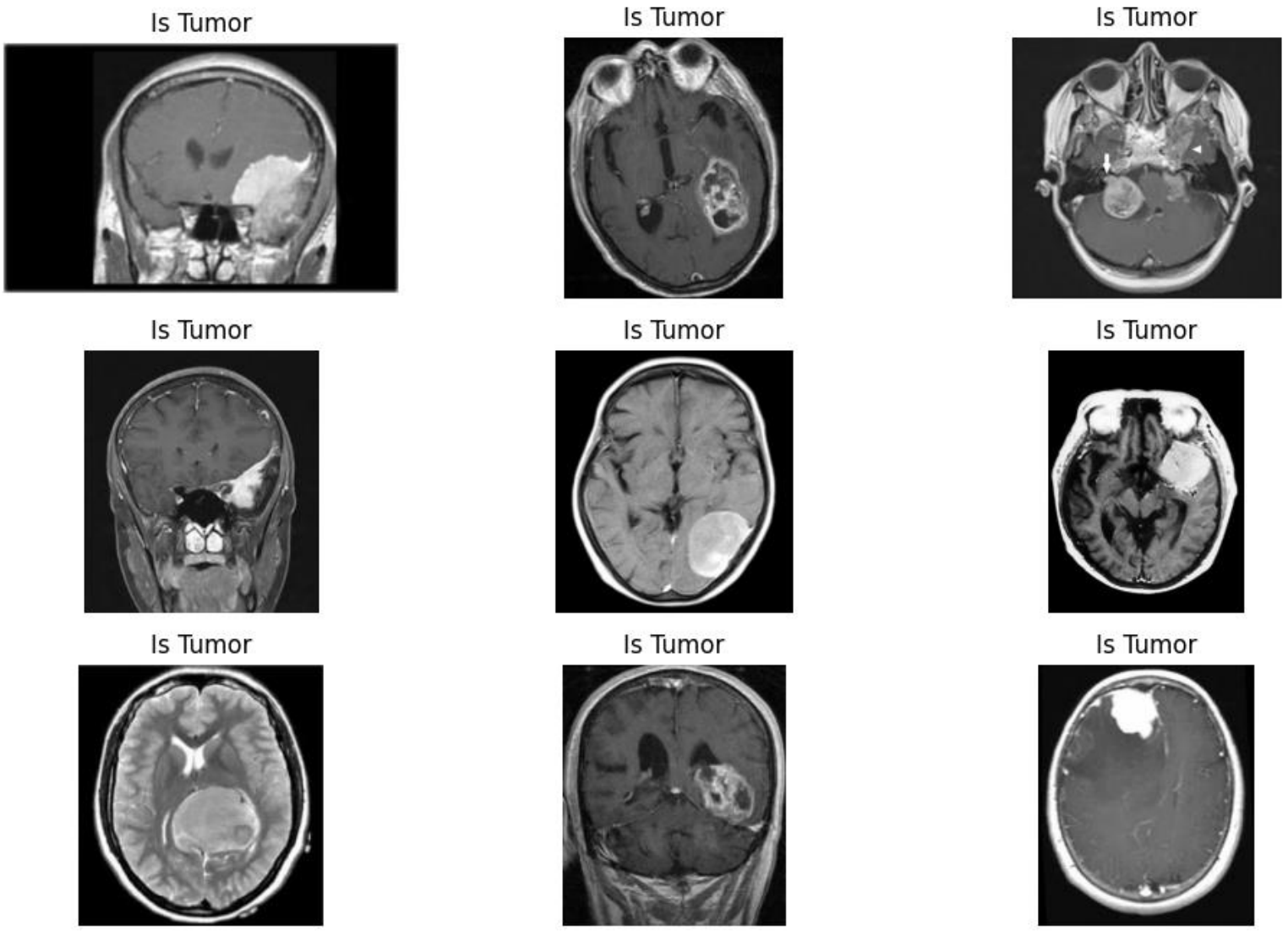
```
[12]: test_based_on_images(classifier, 'pituitary_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Meningioma Tumor Detection – Random Forest Classifier

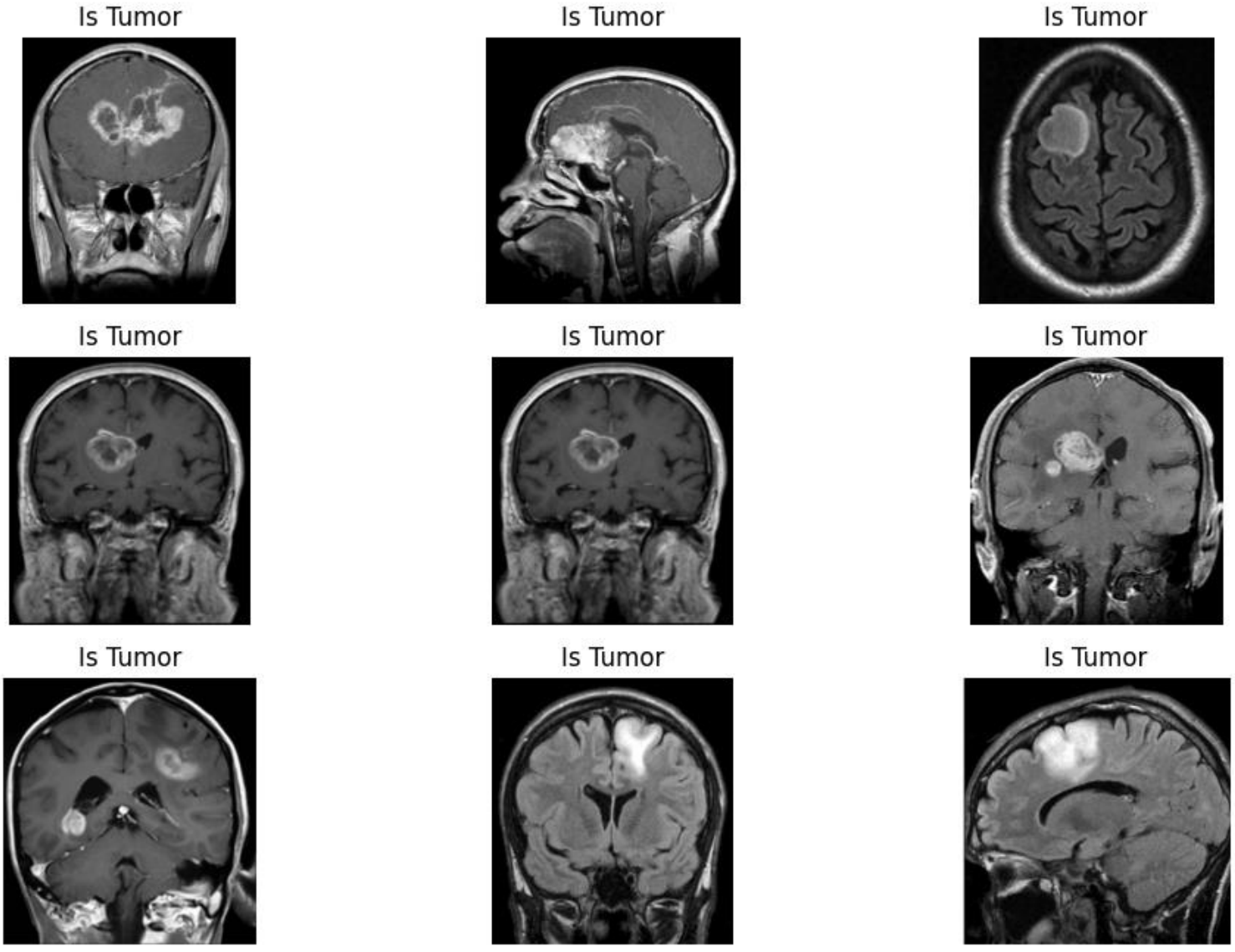
```
[13]: test_based_on_images(classifier, 'meningioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Glioma Tumor Detection - Random Forest Classifier

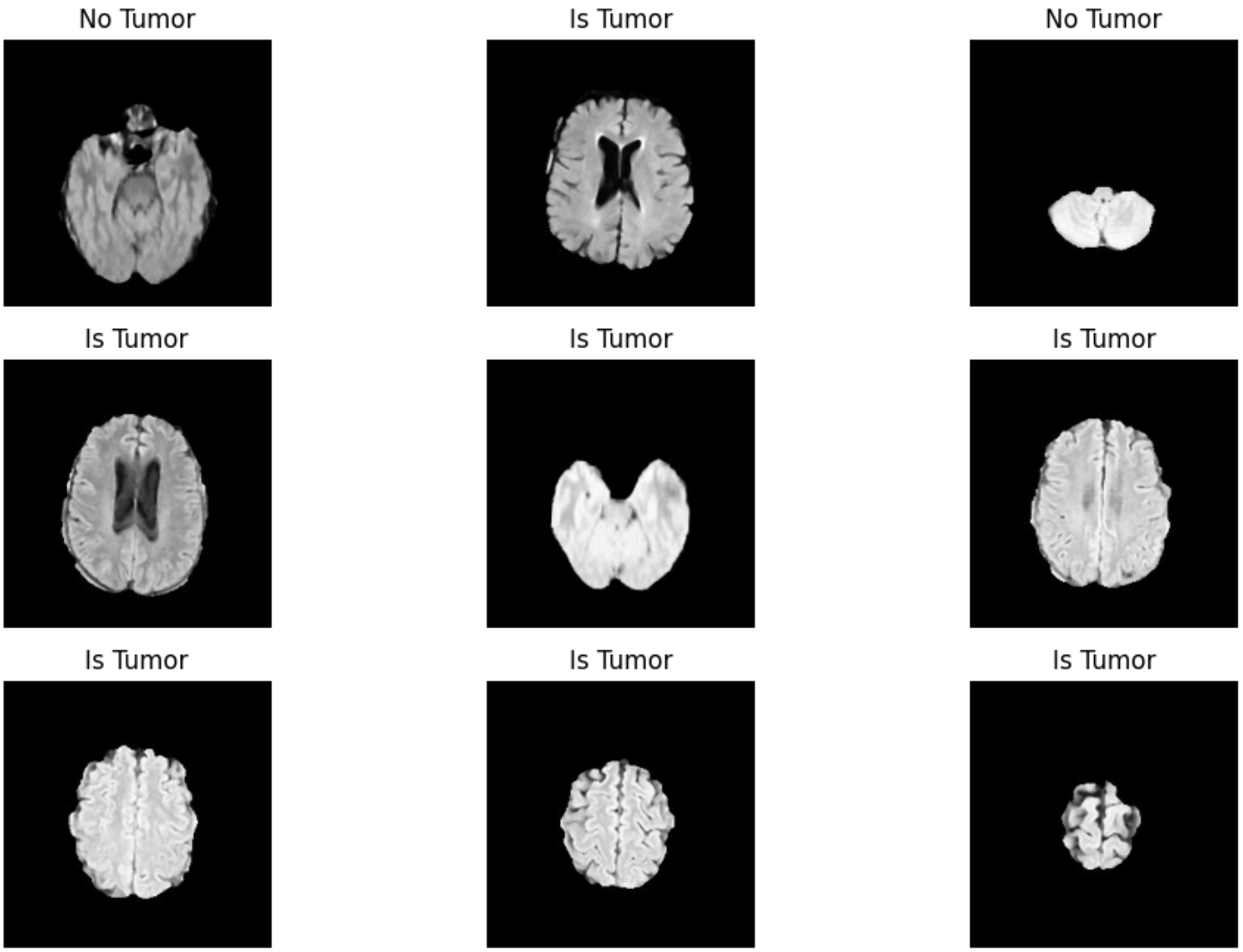
```
[14]: test_based_on_images(classifier, 'glioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

New Tumor Dataset Detection – Random Forest Classifier

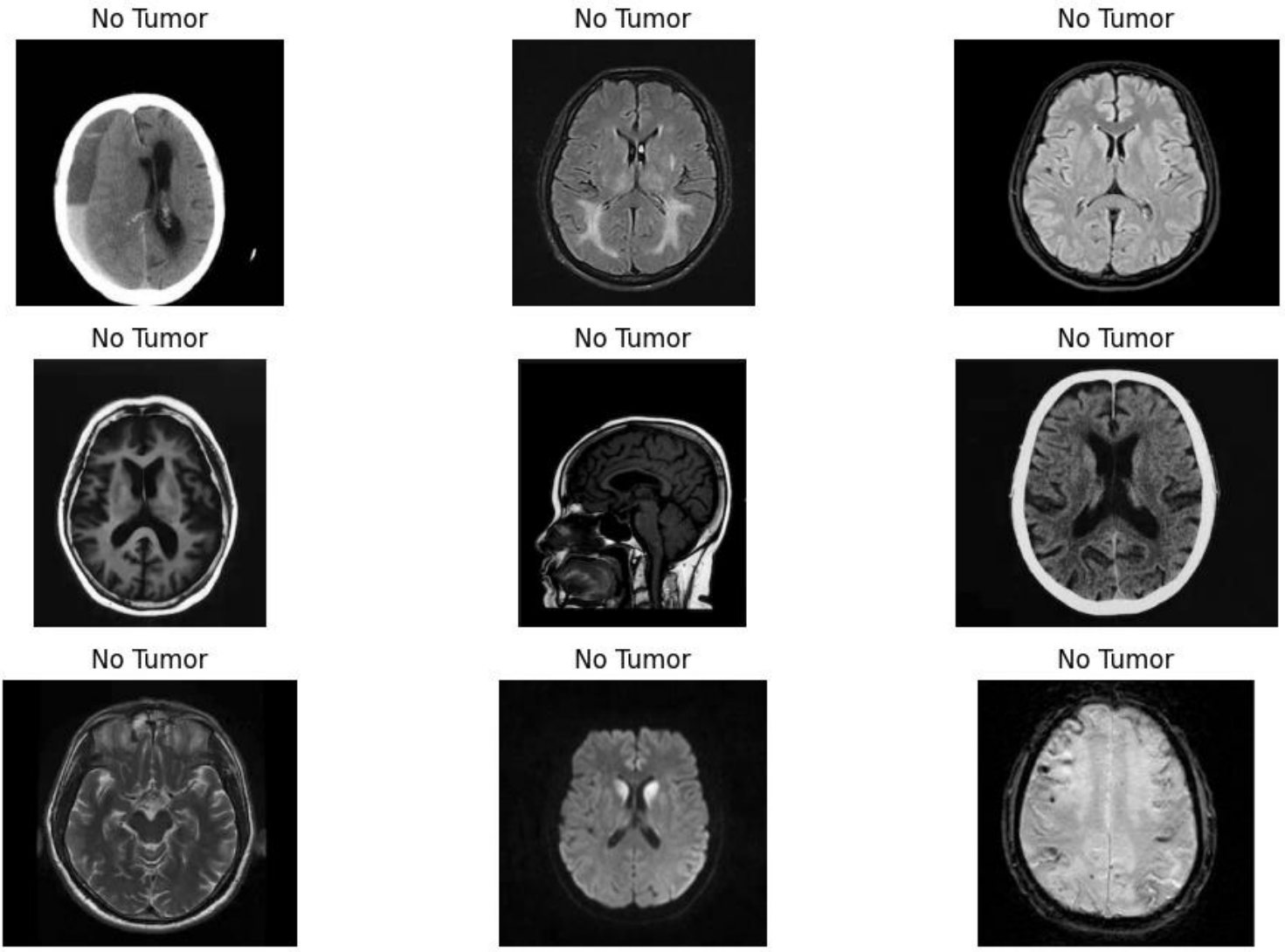
[15]:
```
test_based_on_images(classifier, 'brain_tumor/', 20)
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

No Tumor – Support Vector Machine – Support Vector Classifier:

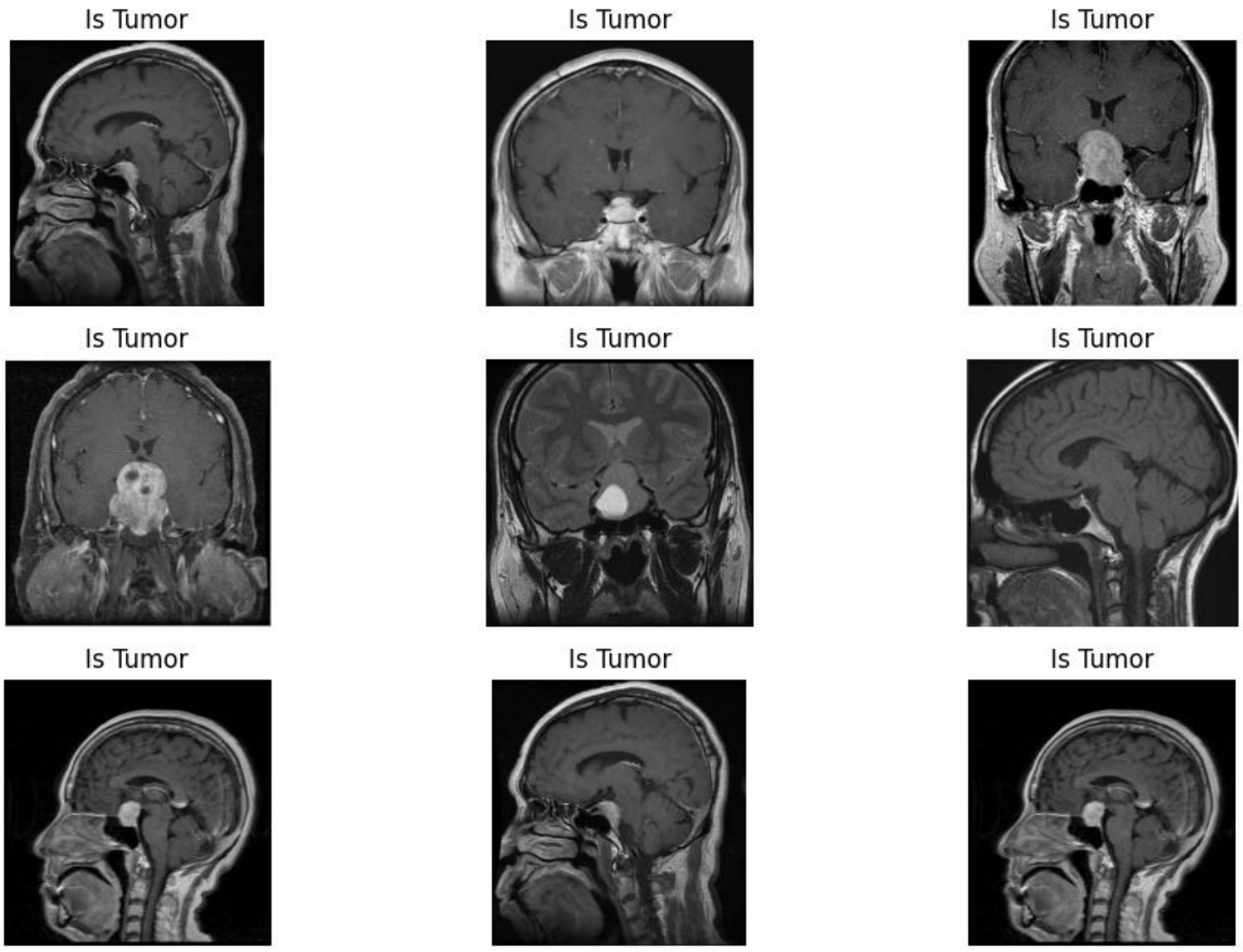
```
[11]: test_based_on_images(classifier, 'no_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

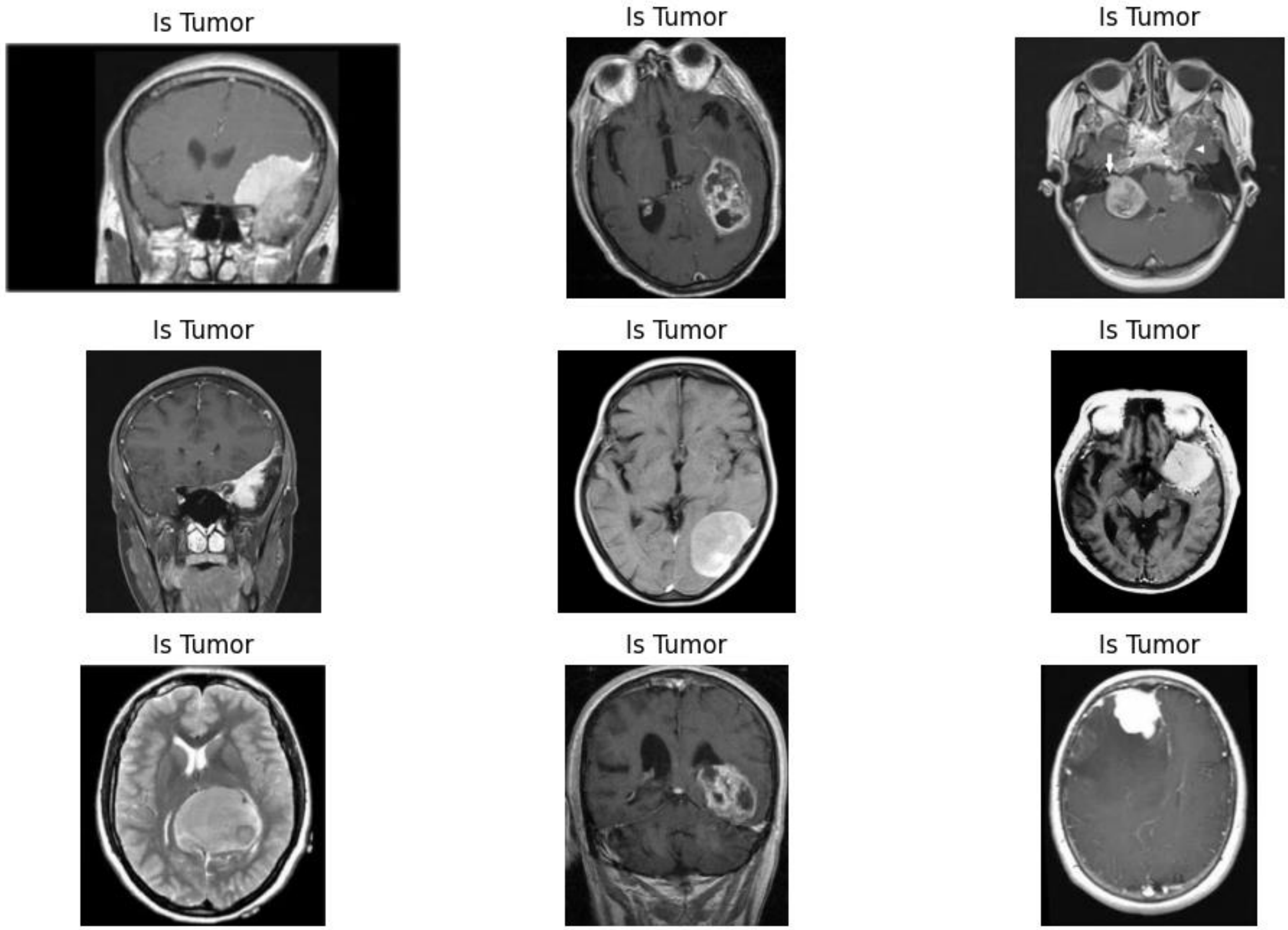
Pituitary Tumor Detection – Support Vector Machine – Support Vector Classifier:

```
[12]: test_based_on_images(classifier, 'pituitary_tumor/')
```



Meningioma Tumor Detection – Support Vector Machine – Support Vector Classifier:

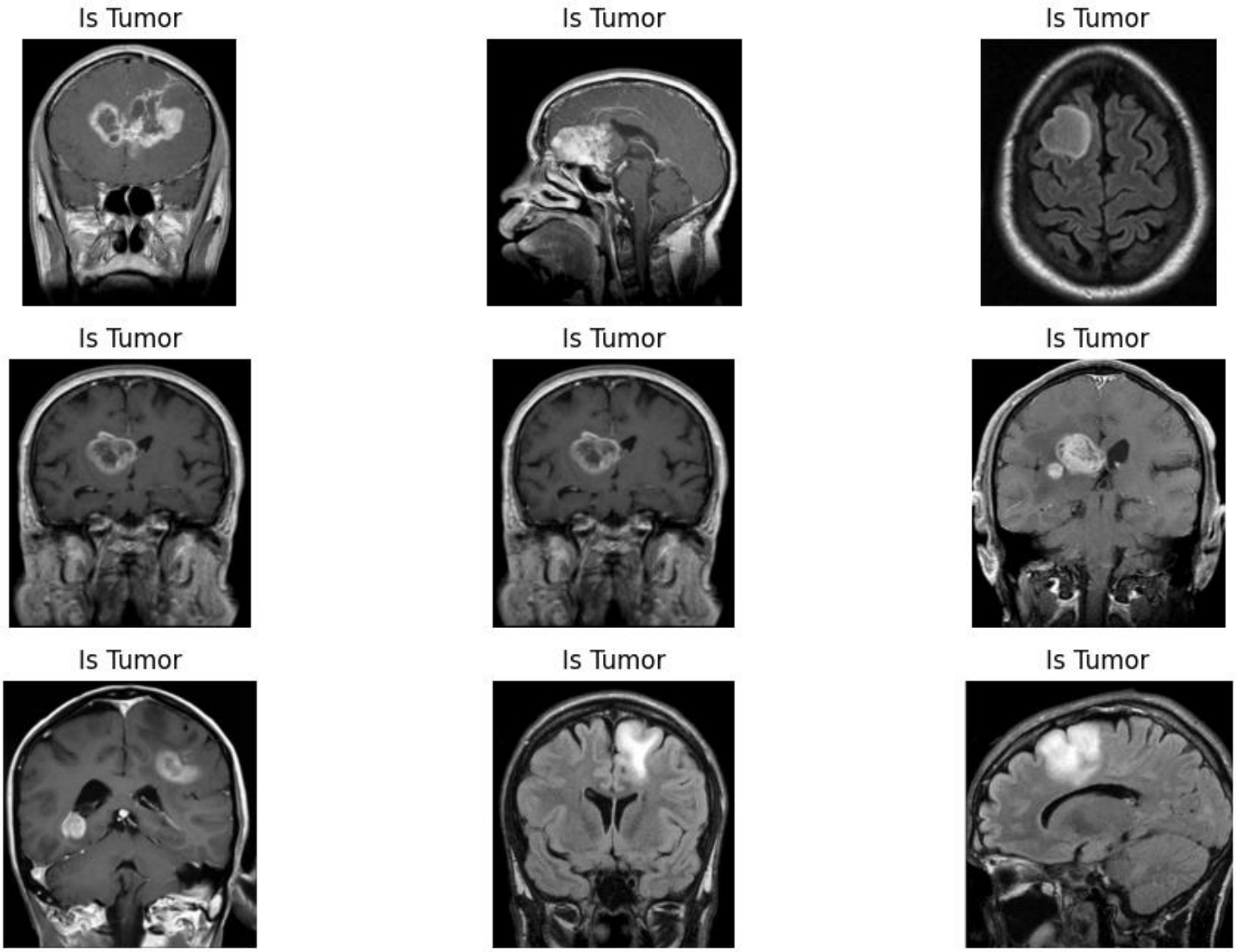
```
[13]: test_based_on_images(classifier, 'meningioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

Glioma Tumor Detection – Support Vector Machine – Support Vector Classifier:

```
[14]: test_based_on_images(classifier, 'glioma_tumor/')
```



PROIECT INGINERIA REGLARII AUTOMATE II					
NUME student		GRUPA:		Nota	

New Tumor Detection Dataset – Support Vector Machine – Support Vector Classifier

```
[15]: test_based_on_images(classifier, 'brain_tumor/', 20)
```