*Nicolae Constantin-Eduard*

*Grupa 244*

# Brain Anomaly Detection

1. Descriere proiect

Proiectul presupune clasificarea unor imagini reprezentând radiografii ale creierului uman, de dimensiune 224x224 pixeli în două categorii(clase), după cum urmează: clasa 1 reprezintă radiografiile care prezintă anomalii(ale unui creier bolnav), iar clasa 0 reprezintă radiografiile normale(ale unui creier sănătos).

2. Setul de date

Am primit ca și input pentru rezolvarea cerinței un număr de 22149 de imagini reprezentând radiografii ale creierului uman, acestea fiind împărțite astfel:

-15 mii de imagini au reprezentat datele de antrenament, pe care le-am folosit pentru a antrena modelele folosite în rezolvarea cerinței

-două mii de imagini au reprezentat datele de validare, pe care le-am folosit pentru a obține acuratețea pe care modelul ales a reușit să o returneze

-5149 de imagini au reprezentat datele de test, pe care le-am folosit pentru a putea genera și completa fișierul de output, în formatul precizat în cerință, cu ajutorul modelului antrenat pe datele precizate anterior

3. Modalități de rezolvare a cerinței și algoritmi utilizați

Pentru rezolvarea cerinței precizate mai sus am folosit două modele: Naive Bayes și Random Forest, abordări pe care le voi prezenta în cele ce urmează.

*a) Metoda folosind Naive Bayes*

Naive Bayes este un clasificator care se bazează pe teorema lui Bayes, care ne oferă probabilitatea valorii de adevăr a unei ipoteze, judecând după dovezile aduse în prealabil. În contextul cerinței curente, algoritmul poate fi folosit pentru a calcula probabilitatea unei clase, ținând cont de datele primite. Începând implementarea algoritmului, inițial am citit label-urile de antrenament și de validare din fișierele aferente (train_labels.txt și validation_labels .txt) și le-am stocat în câte un array(train_labels și val_labels), pentru ca mai apoi să trec la citirea și procesarea imaginilor din folder-ul data. Așadar, nefiind garantat faptul că imaginile din folder vor fi citite și stocate în lista image_files în ordinea necesară pentru rezolvarea cerinței, am decis că aceasta va trebui să fie sortată, pentru siguranța funcționării algoritmului. Odată sortate, imaginile trebuie procesate, așa că am optat pentru a-mi crea o funcție care să proceseze imaginile câte una pe rând, eficientizând astfel algoritmul din punctul de vedere al memoriei ocupate în timpul execuției. Astfel, imaginea curentă va fi inițial transformată în grayscale, apoi va fi adusă la dimensiunea necesară, adică 224x224 pixeli, pentru ca, ulterior, aceasta să fie transformată într-un array. Înainte de a scoate imaginea din memorie, am decis că datele acesteia ar trebui normalizate și am implementat acest lucru. În continuare, pentru o accesare mai facilă a imaginilor câte una pe rând, am definit o funcție image_generator care parcurge lista de imagini și apelează funcția read_image definită mai sus pentru procesarea unei imagini. Imaginile citite și procesate cu ajutorul acestor două funcții vor fi stocate într-un array numit all_images. Știind deja că imaginile au fost sortate înainte de a fi procesate, le putem împărți direct în trei părți, și anume: imaginile de antrenament(imaginile de la 0 la 15000), imaginile de validare(imaginile de la 15001 la 17000) și imaginile de test(imaginile de la 17001 la 22149). În continuare, imaginile vor fi transformate în tablouri bidimensionale întrucât modelul Naive Bayes necesită acest tip de date pentru buna sa antrenare și mai apoi validare și testare, iar mai apoi acestea vor fi normalizate cu ajutorul clasei StandardScaler din librăria sklearn.preprocessing.

În continuare, modelul Naive Bayes va fi antrenat pe datele de antrenament folosindu-ne de clasa GaussianNB din cadrul librăriei sklearn.naive_bayes. Modelul va fi antrenat în mod continuu pe date de câte 1000 de imagini până când se vor termina cele 15 mii de imagini de antrenament. După ce modelul va fi antrenat, el va fi evaluat pe datele de validare, iar mai apoi acesta va fi utilizat pentru a crea fișierul de output, în forma cerută în cerință, fiecare linie a acestuia având două coloane, prima conținând id-ul imaginii respective, scris sub forma prezentată ca și exemplu în cerință, iar a doua coloană reprezentând predicția clasei imaginii de către modelul Naive Bayes antrenat.

*b) Metoda folosind Random Forest*

Algoritmul Random Forest se bazează pe ideea de a construi un ansamblu de arbori de decizie(un ,,forest") și de a combina predicțiile lor pentru a ajunge la o predicție finală. El funcționează prin împărțirea setului de date de antrenament în mai multe subset-uri alese în mod aleatoriu, pentru fiecare dintre acestea fiind construit un arbore de decizie. Această metodă de construire a unui ansamblu de arbori de decizie permite clasificatorului Random Forest să obțină o acuratețe a predicției mai mare decât ar putea obține un singur arbore de decizie, datorită abilității sale de a reduce variația și de a preveni supraînvățarea.

Pentru rezolvarea cerinței date, am început prin a citi label-urile de antrenament și cele de validare din fișierele aferente(train_labels.txt și validation_labels.txt) și stocarea acestora în câte un array separat(train_labels și validation_labels). Ulterior, am separat datele din coloana class ale array-urilor de antrenament și validare în doi vectori separați, pentru o mai ușoară accesabilitate. Am parcurs același proces și pentru label-urile de test(validare) și le-am stocat în array-ul sample_submission.

În continuare, am trecut la citirea imaginilor din folder-ul data, a cărui cale am stocat-o în variabila data_dir. Am inițializat, de asemenea, variabila image_size cu valorile necesare ale dimensiunii unei imagini(224x224 pixeli). Ulterior, ca și în metoda prezentată mai sus(folosind Naive Bayes), am optat pentru definirea unei funcții(preprocess_image) care va citi și va procesa fiecare imagine din folder individual. Astfel, în cadrul funcției, imaginea va fi citită din folder, va fi adusă la dimensiunea image_size și va fi adusă la forma unui tablou unidimensional. Mai departe, vom defini o funcție(preprocess_images) care va stoca imaginile într-un vector, folosindu-se de apeluri utile ale funcției preprocess_image definită mai sus.

Odată stocate într-un vector, imaginile trebuie sortate pentru a putea fi împărțite în mod corect în imagini de antrenament, imagini de validare și imagini de test(validare). Imaginile de antrenament și cele de validare vor fi stocate, ulterior sortării și împărțirii, în vectorii train_images și val_images.

În continuare, modelul Random Forest va fi antrenat pe datele de test, iar ulterior antrenării sale, îi vom testa acuratețea pe datele de validare. În partea finală a programului, vom procesa imaginile de test cu ajutorul celor două funcții descrise mai sus, iar, mai apoi, vom folosi modelul Random Forest antrenat pentru a genera predicțiile pe setul de date de test. Mai departe, vom crea fișierul de submisie, în aceeași manieră cum am procedat și la abordarea cu modelul Naive Bayes, singura diferență dintre cele două generări a fișierului de submisie fiind aceea că la abordarea cu Naive Bayes predicțiile au fost făcute individual

pentru fiecare dintre imagini în timpul parcurgerii lor, în timp ce la abordarea cu Random Forest predicțiile au fost făcute anterior și stocate în vectorul test_predictions.

4. Referințe


- cod laborator
- https://fmi-unibuc-ia.github.io/ia/
- https://scikit-learn.org/stable/index.html