Brain Anomaly Detection

Bălescu Alexandru

14/04/2023

Contents

1 Introducere & Descrierea proiectului

Acest proiect a constant în clasificarea de imagini monochrome a tomografiilor la creier. Acestea puteau sa fie încadrate în 2 clase, iar dimensiunea lor este de 224x224 pixeli. Pentru rezolvare am folosit mai mulți clasificatori, dar în continuare voi prezenta modelele bazate pe Naive Bayes, Random Forests si CNN.

2 Naive Bayes

Unul dintre clasificatorii pe care i-am încercat este Gaussian Naive Bayes. Acesta este un tip de clasificator probabilistic folosit pentru a clasifica obiectele în două sau mai multe categorii. Principiul de funcționare se bazează pe folosirea teoremei lui Bayes pentru a calcula probabilitățile condiționate ale fiecărei categorii, date un set de caracteristici ale obiectului care trebuie clasificat.
Formula, unde σ_y și μ_y sunt estimați folosind probabilitatea maximă:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Pentru citirea datelor, folosesc modulul OpenCV, urmând ca după ce redimensionez imaginile la dimensiunea de 64x64 sa apelez funcția array din numpy ca sa obțin array-uri de 64x64. *Trăsăturile* datelor sunt date de pixelii imaginilor. Prima modificare a fost aceea de a liniariza datele, folosind functia flatten, transformandu-le in array-uri unidimensionale.

Pentru implementare, am folosit clasificatorul GNB din modulul naive_bayes, libraria sklearn. Implementare din sklearn se folosește de formula de mai sus ca sa faca clasificările, totusi, dezavantajul metodelor Naive Bayes este ca nu au cele mai bune estimări, dar sunt destul de eficiente din punct de vedere al vitezei.

3 Random Forests

O alta abordare este cea de a folosi Random Forests. Acesta este un tip de clasificator cu învățare supervizată, fiind antrenat pe un set de date etichetate. Algoritmul Random Forests se bazează pe un ansamblu de arbori de decizie (decision trees). Fiecare arbore de decizie este construit pe baza unui subset al setului de date de antrenare, ales aleatoriu și cu înlocuire, și pe baza unui set aleatoriu de caracteristici (features) pentru a realiza predicții. Apoi, ansamblul de arbori de decizie este folosit pentru a face predicții asupra datelor noi, prin combinarea predicțiilor făcute de fiecare arbore de decizie individual.

Pentru citirea datelor, folosesc modulul OpenCV, urmând ca după ce redimensionez imaginile la dimensiunea de 64x64 sa apelez funcția array din numpy ca sa obțin array-uri de 64x64. *Trăsăturile* datelor sunt date de pixelii imaginilor. Prima modificare a fost aceea de a liniariza datele, folosind functia flatten, transformandu-le in array-uri unidimensionale.

Pentru implementare, am folosit clasificatorul RandomForestClassifier din modulul ensemble, libraria sklearn. Implementarea din sklearn se folosește de o serie de parametrii care influențează modul în care funcția funcționează. Parametrii folosiți sunt:

- n_estimators = 200 : antrenarea se va face pe 200 de arbori (cu cat e mai mare e mai bine, dar durează mai mult computarea și după un anumit număr de arbori, rezultatul nu se mai imbunatateste significant)
- max_features = 0.5 : la fiecare nod se vor lua doar 50% din caracteristici(cu cat e mai mic se reduce varianta, dar creste bias-ul)
- max_depth = 20 : la fiecare nod o sa fie 20 de nivele
- n_jobs = -1 : folosește toate nucleele calculatorului pentru antrenare

4 Convolutional Neural Network

O ultima abordare pe care am incercat-o este aceea de a lucra cu un CNN. Libraria pe care am folosit-o este Tensorflow, utilizand modulul Keras. Am inceput prin a-mi procesa datele

folosind modulul OpenCV, am transformat imaginile in format monochrom folosind IMREAD_GREYSCALE si le-am transformat in array-uri de 224x224. Ulterior am transformat label-urile in format ‘float64’ca sa nu existe erori la antrenare.

Am creat un model de rețea neuronala fără niciun layer folosindu-ma de functia Sequential din Keras. Ulterior am adaugat manual layere folosindu-ma de metoda add.

Primul layer are rolul de a pre procesa datele: argumentul “scale” specifica factorul de scalare. Am ales valoarea de 1./255 pentru a normaliza pixeli din intervalul [0,255] la intervalul [0,1] ca sa imbunatatesc timpul de executie. Argumentul “input_shape” specifica dimensiunea imaginilor de intrare(224x224 pixeli) si un singur canal de culoare(am transformat anterior imaginile in format grayscale).

Am 3 layere de tip Conv2D. Mai precis, prin intermediul clasei Conv2D a modulului layers, această linie definește un strat de convoluție care va fi adăugat în rețeaua neuronală. Argumentul "32" specifică numărul de filtre (sau canale de ieșire) care vor fi generate de acest strat de convoluție. Fiecare filtru aplică o matrice de convoluție (kernel) de dimensiunea (5, 5) asupra imaginii de intrare pentru a extrage caracteristici din imagine.
Argumentul "activation" specifică funcția de activare care va fi aplicată rezultatelor de ieșire ale stratului de convoluție, în acest caz "relu" (rectified linear unit). Funcția de activare relu este una dintre cele mai comune funcții de activare utilizate în rețelele neuronale convoluționale, deoarece poate ajuta la evitarea problemei de dispariție a gradientului și poate spori viteza de convergență a rețelei.

Am 3 layere de tip MaxPooling2D. Mai precis, prin intermediul clasei MaxPooling2D a modulului layers, această linie definește un strat de Pooling care va fi adăugat în rețeaua neuronală.
Argumentul "pool_size" specifică dimensiunea ferestrei de Pooling. Stratul de Pooling reduce dimensiunea imaginii, luând valoarea maximă din fiecare fereastră de dimensiunea (3, 3) a imaginii de intrare. Scopul acestui strat este de a reduce dimensiunea imaginii, ceea ce ajuta la evitarea problemei de overfitting.

Am 3 layere de tip Dropout. Acestea au rolul de a renunta in mod aleator la trasaturi invatate anterior, reducand riscul de overfitting. Argumentul de ”0.2” si “0.3” înseamnă că 20%, respectiv 30% dintre ieșirile stratului anterior vor fi setate la zero în timpul antrenării.

Ultimele 3 layere pe care le-am adaugat au rolul de a clasifica imaginile:

model.add(Flatten()): Această linie adaugă un strat de aplatizare (Flatten) în rețea. Acest strat transformă ieșirile tridimensionale ale straturilor anterioare într-un vector unidimensional, astfel încât acestea să poată fi procesate de un strat complet conectat (Dense).

model.add(Dense(8, activation='relu')): Această linie adaugă un strat tip Dense la rețeaua neuronală cu 8 neuroni. Acesta utilizează funcția de activare relu pentru a introduce non-linearitate în rețeaua neuronală.

model.add(Dense(1, activation='sigmoid')): Această linie adaugă un strat complet conectat final cu un singur neuron, care e responsabil pentru clasificarea imaginilor în două clase.

Stratul utilizează funcția de activare Sigmoid, care transformă ieșirile într-un interval de [0, 1], interpretate ca probabilități pentru cele două clase.

Ulterior am folosit un optimizer si am compilat codul:
Am folosit optimizerul Adam, care este o metodă populară de optimizare a gradientului stocastic.
Argumentul learning_rate specifică rata de învățare a optimizerului (în acest caz, 0.0001), iar argumentul amsgrad specifică dacă se utilizează varianta AMSGrad a optimizerului Adam.

Am folosit functia compile() pentru a compila codul .Aceasta primește trei argumente:

optimizer: optimizerul folosit pentru antrenarea modelului, în acest caz, Adam;
loss: funcția de loss folosită pentru evaluarea performanței modelului în timpul antrenării, în acest caz, binary_crossentropy, care este folosită pentru problemele de clasificare binară;
metrics: metricile de performanță folosite pentru evaluarea modelului în timpul antrenării și testării, în acest caz, metrica de acuratețe este utilizată pentru a evalua accuratetea modelului.
Ulterior am antrenat modelul pe datele de train si validation, folosind 20 de epoci, apoi am comparat rezultatele fata de 0.45(empiric obtinut) ca sa obtin o clasificare cu 0 sau 1.

5 Concluzii

Am trecut de baseline-ul cu toate modelele, cele mai bune fiind Random Forest (aprox 0.46) si CNN (aprox 0.61)