**Proiect - Inteligență artificială**

**Anul 2022-2023**

**Neaga-Budoiu Maria, grupa 232**

Acest proiect a avut ca scop rezolvarea problemei Deep Hallucination Classification, pentru care am folosit două modele de clasificare a imaginilor primite: KNN și Naive-Bayes. Mai jos, am explicat etapele de citire și de scriere a datelor, comune ambelor implementări, apoi am descris în particular fiecare model.

Citirea și păstrarea datelor

Am citit întâi numele imaginilor și etichetele corespunzătoare pentru datele de antrenare și testare, precum și numele imaginilor de test, folosind funcția reader() din librăria csv. Am convertit etichetele la tipul int înainte de a le stoca în listele corespunzătoare. Dupa aceea, am citit imaginile efective, pastrându-le în array-uri din biblioteca numpy, dar nu înainte de a condensa imaginile, folosind funcția flatten() pentru a le reduce dimensiunea. Modelul KNN și cel Naive Bayes operează pe array-uri de tablouri unidimensionale, adică dimensiune 2, iar imaginile sunt bidimensionale, deci array-ul de imagini ar avea 3 dimensiuni dacă nu s-ar aplica condensarea.

Afișarea predicțiilor

Având predicțiile pentru imaginile de test deja calculate, am deschis fișierul de output (care va fi creat în cazul în care nu este existent) în modul de scriere, apoi am folosit writer din librăria csv, afișând fiecare rând în parte cu writerow().

**Modelul KNN**

Pentru a rezolva problema de clasificare Deep Hallucination, am utilizat modelul KNN, prezentat și în laboratorul 3. Pe scurt, metoda celor mai apropiați vecini alege un număr de vecini, k, precum și o măsură a distanței. După calcularea distanțelor față de toate datele de antrenament pentru o instanță, se aleg primele k cele mai apropiate dintre aceste date. Eu am ales distanța ***hamming*** și **7** vecini, după mai multe încercări efectuate pe datele primite.

Normalizarea

Înainte de a aplica modelul, am normalizat atât datele de antrenare, cat și pe cele de validare și testare folosind normalizare de tipul MinMax (from skelarn.preprocessing import MinMaxScaler). În cazul în care aș fi aplicat normalizarea doar pe imaginile de antrenare, etichetarea s-ar fi făcut pe un anume format de date, pe când validarea și testarea pe unul diferit, ceea ce ar fi dus la o clasificare incorectă.

MinMaxScaler este tehnica de scalare care prelucrează date astfel încât ele să fie toate cuprinse într-un interval. Formula care stă la baza sa este: normalized\_data = (data – min) / (max – min), ceea ce asigură încadrarea în intervalul min-max, de obicei consierat 0-1.

Am normalizat datele după efectuarea unor încercări: varianta fără normalizare obținea o acuratețe de 0.19 pe datele de validare cu folosirea în model a 7 vecini și a metricii *hamming*, pe când, în aceleași condiții, dar cu normalizarea MinMaxScaler, acuratețea rezulată este de 0.196.

Definirea modelului

Pentru implementare, am utilizat KNeighboursClassifier din biblioteca sklearn, modulul neighbours. Am antrenat modelul prin aplicarea funcției fit(), după care am obținut predicțiile pe mulțimea de validare și de test cu metoda predict(). Pentru stabilirea acurateței, am făcut media numărului de etichete corect prezise, folosind funcția mean() din numpy.

Am ales metrica *hamming*, după ce am încercat și cu cea euclidiană și Manhattan, obținând acuratețe mai slabă. Mai jos, se află un grafic al variației măsurii de acuratețe pentru numere de vecini diferite și metricile enumerate anterior, rezultate obținute în urma încercărilor.

A graph of a number of neighbors

Description automatically generated with low confidence

Am obținut precizia folosind funcția precision\_score, cu parametrul average setat la valoarea ‘weighted’ pentru a calcula independent precizia pentru fiecare label și a obține apoi o valoare medie. Am încercat, de asemenea, să nu specific nicio valore pentru parametrul average, ceea ce a dus la o afișare a preciziilor separate pentru toate etichetele. În plus, am adăugat zero\_division = 0.1 astfel încât să evit eroarea în urma împărțirii la 0. Precizia afișată a fost de 0.3822616147632589.

Pentru matricea de confuzie, am ales la întâmplare câteva etichete de clase din setul val\_labels, pentru a nu afișa o matrice foarte mare, cu toate etichetele. Astfel, am format o submatrice din cea inițială cu ajutorul indicilor etichetelor alese și a verificării np.isin(knn.classes\_, chosen\_labels), pe care am plotat-o folosindu-mă de clasa DisplayConfusionMatrix din sklearn.metrics. Valorile pozitive din matrice reprezintă instanțele corect prezise pentru un anume label.

A picture containing screenshot, text, square, diagram

Description automatically generated

**Modelul Naive Bayes**

Al doilea model utilizat este Naive Bayes, un clasificator care se bazează pe probabilități condiționate (Teorema lui Bayes). Ținând cont de aparițiile etichetelor de clase în datele de antrenare, acesta construiește un model probabilistic pe care îl va utiliza pentru datele de test și validare.

Normalizarea

Tipul de normalizare folosit este identic cu cel regăsit în implementarea modelului KNN. Nu am obținut o diferență de acuratețe pe datele de validare atunci când am aplicat prenormalizarea, dar am păstrat-o întrucât precizia a avut o valoare mai bună.

Definirea modelului

După importarea modului naive\_bayes din biblioteca sklearn, am implementat modelul cu ajutorul clasei GaussianNB. Am ales-o pe aceasta în defavoarea clasei MultinomialNB deoarece am obținut o acuratețe mai bună în urma încercărilor, anume 0.27 în loc de 0.186. Este de menționat și faptul că MultnomialNB funcționează pe valori discrete atât ale imaginilor de antrenare, cât și ale etichetelor pentru acestea, dar valorile label-urilor de antrenare se repetă în cazul acestui proiect. În următorul tabel, se pot observa diferențele dintre rezultatele celor două modele.

|  |  |  |  |
| --- | --- | --- | --- |
| **Modelul** | **Acuratețe fără normalizare** | **Acuratețe după normalizare** | **Precizie** |
| GaussianNB | 0.27 | 0.27 | 0.3189185854885442 |
| MultinomialNB | 0.184 | 0.186 | 0.20863299823007733 |

În următoarea imagine, se află o porțiune aleasă aleator din matricea de confuzie:

A picture containing screenshot, square, rectangle, colorfulness

Description automatically generated

Concluzie

Deși rezultatele implementărilor sunt apropiate în ceea ce privește acuratețea, se observă totuși că versiunea modelului Naïve Bayes conferă o acuratețe mai bună pentru datele de validare decât modelul KNN.