Facultatea de Matematică și Informatică

Universitatea din București

Proiect – Automatic Misogyny Identification

Inteligență artificială

Student, Marin Constantin – Cătălin,

Grupa 354

Cuprins:

Clasificatorul Multinomial Naïve Bayes

Caracteristici folosite

În primul rând, pentru a putea reprezenta tweet-urile noastre într-un mod eficient pentru clasificare, vom folosi reprezentarea 'Bag of Words' cu primele 1000 de cuvinte alese după numărul de apariții.

Pentru aceasta, în primul rând, am preprocesat datele, atât de antrenare, cât și de test, prin intermediul funcției tokenize(text). Astfel, în cadrul acesteia am transformat toate caracterele în caractere mici, am eliminat whitespace-urile, link-urile, cuvintele care încep cu @ sau #, numerele și cuvintele cu o dimensiune mai mică decât 3. De asemenea, am folosit și ItalianStemmer, lemmatizer, eliminarea stopwords-urilor, dar nu au oferit o performanță bună în ceea ce privește f1_score-ul așa că le-am comentat. Pentru tokenizare, am folosit RegexpTokenizer, care separă cuvintele în funcție de orice caracter care nu se regăsește în '[a-zA-Z0-9_]'.

Parametri, hiperparametri

Hiperparametrul acestui model îl reprezintă 'alpha' care este un număr real cu valoarea default egală cu 1 și reprezintă un hiperparametru aditiv de liniarizare(Laplace, alpha=1, sau Lidstone, alpha<1), care când este mai mare sau egal cu 0 are rolul de a preveni valorile de zero pentru probabilitățile calculate.

Probabilitatea likelihood, notată P(x|y), adică probabilitatea ca un cuvânt să se afle într-un sample ce-i aparține clasei y este egală cu raportul dintre numărul de apariții ale cuvântului în clasa y + alpha și numărul de apariții ale cuvântului în toate clasele, corpus + alpha*dimensiunea vocabularului.

'fit_prior' este un parametru de tip bool cu valoarea implicită 'True' care este folosit pentru ca algoritmul să învețe probabilitățile prior ale unei clase sau nu.

'class_prior' este un parametru asemănător unei matrici de forma shape(n_classes) cu valoarea default egală cu None, unde n_classes reprezintă cele 2 clase, 0 sau 1. Aceasta reține probabilitățile prior ale clasei.

Antrenarea parametrilor/hiperparametrilor

Pentru antrenarea parametrilor/hiperparametrilor am folosit 10 fold cross-validation pe care l-am inclus într-un for cu 10 pași, numiți epoci, în care am calculat media f1_score-urilor, deviația standard, matricea de confuzie. De asemenea, am calculat pentru fiecare pas din 10 fold, matricea de confuzie și f1_score-urile.

În continuare, m-am orientat după numarul de 1 și de 0 din cadrul predicțiilor și pe baza rezultatelor obținute pe platforma Kaggle și am observat că cele mai bune rezultate le-am obținut folosind valorile default atât pentru parametri, cât și hiperparametri.

Durata antrenării

Pentru a calcula durata antrenării vom folosi librăria time. Astfel, am decis să calculez timpul de antrenare atât pe bucata de clf.fit, cât și pe întregul for care conține tot procesul de 10 fold cross-validation.

Pentru prima variantă, definim o listă, timp_antrenare_fold, vidă, și două variabile t_start_antrenare_fold, înainte de clf.fit, și t_sfarsit_antrenare_fold, după clf.fit. Ambele vor primi valoarea time.time() care returnează timpul exprimat în secunde ca număr în virgulă mobilă și apoi vom face diferența dintre t_sfarsit_antrenare_fold și t_start_antrenare_fold pe care o vom adăuga în lista noastră, iar după ce se termină for-ul respectiv, vom afișa atât media timpilor de antrenare, cât și suma timpilor de antrenare și obținem: „**media timpilor de antrenare: 0.014560174942016602 si timpul total de antrenare pe clf.fit in cadrul 10 fold: 0.14560174942016602"**.

Pentru a doua variantă, vom defini cele două variabile de mai sus, în afara for-ului cu 10 fold, t_start_antrenare_fold, înainte și t_sfarsit_antrenare_fold, după, iar la final vom face diferența dintre cele două și o vom afișa: „**timpul de antrenare: 0.4849221706390381"**.

Performanța de pe Kaggle pe 40% date din set

Utilizând acest clasificator alături de cele menționate anterior, am obținut pe datele publice din cadrul concursului de pe Kaggle **performanța 0.78125**.

Pentru modelul Multinomial Naïve Bayes am ales să folosesc 10 fold cross-validation și am obținut următoarele rezultate în urma antrenării:

- Matricea de confuzie finală, adică suma tuturor matricelor obținute în timpul pașilor din 10 fold:

  „**Matricea de confuzie este:**

  [[2315. 348.]

  [294. 2043.]]”

- Media f1 scorurilor și deviația standard:

„**Media f1 scorurilor este: 0.8640531551515235 cu deviatia standard: 0.014792953930452074”**

Clasificatorul kNN

Caracteristici folosite

La fel ca în cazul modelului Multinomial Naïve Bayes, am folosit reprezentarea ‘Bag of Words’, dar de data aceasta cu primele 100 de cuvinte în funcție de numărul de apariții. Deși mai sus am folosit preprocesarea datelor, aceasta a condus la un f1 score mult mai slab, atât local, cât și pe Kaggle, și astfel, am decis să păstrez numai tokenizarea cu WordPunctTokenizer.

Parametri, hiperparametri

În cadrul acestui model, un hiperparametru îl reprezintă k-ul, care pentru kNN din sklearn este numit ‘n_neighbors’. Acesta este un int, cu valoarea default egală cu 5 și reprezintă numărul de vecini, care sunt cei mai apropiați de data noastră de test, pe care trebuie să o clasificăm.

Un parametru este metrica, având valoarea default ‘minkowski’, cu hiperparametrul p implicit egal cu 2, adică distanța euclidiană, l2.

Parametrul ‘weights’ are ca valoare default ‘uniform’, adică toate punctele din fiecare vecinătate au ponderi egale și este o funcție de calcul a ponderilor folosită în calculul unei predicți.

Parametrul ‘algorithm’ este folosit pentru a alege un algoritm care să calculeze cei mai apropiați vecini și are valoarea default egală cu ‘auto’.

Parametrul ‘leaf_size’ este un int cu valoarea default egală cu 30 care afectează viteza de construire a arborelui și memoria necesară pentru BallTree sau KDTree.

Parametrul ‘n_jobs’ este de tip int, cu valoarea default None și prin intermediul său se specifică numărul de thread-uri folosit.

Antrenarea parametrilor/hiperparametrilor

Spre deosebire de Multinomial Naïve Bayes, am ales să fac antrenarea folosind funcția split, care împarte setul de date în două, 75% date de antrenare și 25% date de validare. Acest split l-am inclus într-un for cu 10 iterații pentru a observa matricea și f1_score-ul pentru fiecare split realizat. În ceea ce privește restul antrenării, am procedat ca la modelul anterior și astfel am obtinut k, n_neighbors egal cu 17, iar metrica și restul valorilor le-am lăsat default. Astfel, algoritmul va alege pe baza distanțelor calculate, primii 17 vecini cu distanțele cele mai mici, și dintre aceștia va selecta eticheta care apare de cele mai multe ori printre ei.

Durata antrenării

Pentru acest model am aplicat aceeași procedură, singura modificare fiind faptul că la kNN am folosit split și am obținut următoarele valori astfel:

- prima variantă: „**durata media a antrenarii acestui model este 0.07664868831634522 si timpul total de antrenare pe clf.fit in cadrul split-ului: 0.7664868831634521”**;
- a doua variantă: „**durata antrenarii acestui model este 7.8479509353637695”.**

Performanța de pe Kaggle pe 40% date din set

Cu ajutorul acestui model și a celor menționate mai sus am reușit să obțin pe setul de date public de pe Kaggle **performanța 0.72795**.

Bibliografie:

- https://stackoverflow.com/questions/54396405/how-can-i-preprocess-nlp-text-lowercase-remove-special-characters-remove-numb
- https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908
- https://stackoverflow.com/questions/7370801/how-to-measure-elapsed-time-in-python