Proiect Inteligență artificială
Complex Word Classification

Gaussian Naïve Bayes

Gaussian Naïve Bayes implementează teorema lui Bayes pentru date care au o distribuție gaussiană. Formula pentru clasificarea cuvintelor:

P(un token sa fie complex | caracteristici)

=

P(token) x P(caracteristici|complex)/P(caracteristici)

Unde:

- *P(token)* = probabilitatea ca atunci când alegem un token să fie complex = probabilitatea apriori
- *P(caracteristici|complex)* = probabilitatea să avem niste caracteristici din care știm că rezultă un token complex, iar acestea să fie exact caracteristicile noastre = probabilitatea likelihood
- *P(caracteristici)* = probabilitatea ca noi să observăm caracteristicile

Modelul Gaussian se diferențiaza de cel de baza prin formula de probabilitate likelihood

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

Caracteristici:

Inițial am filtrat datele propozițiilor prin ștergerea semnelor de punctuație, asigurarea unei tokenizări corecte a cuvintelor și ștergerea cuvintelor de legătura(stop words).

Am folosit un total de 9 feature-uri care îmi oferă complexitatea tokenului:

- *Corpus_features(corpus):* returnează câte un număr caracteristic pentru fiecare corpus în parte;

- *Nr_synsets(word):* returnează numarul de sensuri pe care le are tokenul respectiv cu ajutorul librăriei wordnet;
- *Is_dale_chall(word):* returnează 1 daca tokenul se regăsește în lista de cuvinte Dale- Chall sau 0 dacă nu se regăsește;
- *Is_sentence_complex(sentence):* am considerat că o propoziție este complexă dacă conține mai puțin de 30% de cuvinte Dale-Chall. În funcție de acest procentaj, returnează 1 sau 2;
- *Is_title(word):* returnează 1 dacă tokenul începe cu litera mare, sau 0 dacă începe cu literă mică;
- *Nr_syllabes(word):* returnează numărul de silabe al tokenului;
- *Nr_vowels(word):* returnează numărul de vocale conținute de token;
- *Suffix(word):* returnează 1, 2, 3 sau 4 in funcție de sufixele în care se termină tokenul. Am ales cele mai comune sufixe pentru cuvinte, adjective și verbe, iar în functie de asta am dedus și complexitatea lor. Dacă nu conșine niciunul dintre acestea, se deduce că acest token este mai rar și mai greu;
- *Length_of_word(word):* returnează 1, 2 sau 3, în funcție de lungimea tokenului.

Hyperparametrii funcției sunt lăsați pe valoarea default.

Durată antrenare: 3.2180042266845703s

Scor obținut pe leaderboardul public de pe Kaggle: 0.78237

Matrice de confuzie:[[4954 1958] [188 562]]

Testele cu 10 fold validation:

0.7519246217699826

0.7677937677937678

0.7364206464061747

0.7378146453089245

0.7290303907380608

0.7486543909348442

0.7355775431614247

0.7304057909000143

0.6349275509170129

0.7891215106732348

Media 0.7361670858603441

KNN(K Nearest Neighbors)

Acest algoritm se bazează pe învățarea prin analogie și stabilește clasa unui exemplu de testare pe baza similarității acestuia cu K vecini, cei mai similari, din setul de date de antrenare.

Hiperparametrii:

n_neighbors = 7

Antrenarea hiperparametrilor:

Am ales aceste valori în mod experimental, în urma testelor făcute pe datele de split și pe platforma Kaggle.

Durata antrenare: 89.77685475349426s

Această durată este atât de mare pentru că am adăugat clasificatorul *mean_vector_norm(sentence, word),* în care am calculat media normei vectorului propoziție și vectorul normei tokenului, iar în funcție de diferența între cele două, am returnat 1 sau 2. Acesta funcționeaza cu librăria *spacy* si încarcă din listă „en_core_web_lg", de unde reiese acest timp.

Scor obtinut pe Kaggle public leaderboard : 0.58556