Proiect Inteligență Artificială

Profesor coordonator:

Bogdan Alexe

Sergiu Nișioi

Student

Ulmeanu Cristian

Grupa 353

Cuprins

1. Prezentarea proiectului și a implementării......................................................................3
2. Prezentare și explicare featureruri folosite......................................................................3
3. Clasificatori și rezultate...................................................................................................10

# Prezentarea proiectului si a implementarii

Proiectul prevede implementarea unui algoritm de inteligență artificială ce analizează un set de date de tip csv sub forma: Corpus, Sentence și Token pentru a determina complexitatea tokenului.

Pentru aceasta implementare am avut un set de date de antrenare format din 7661 date, iar pentru testare am avut un set format din 1338 de date.

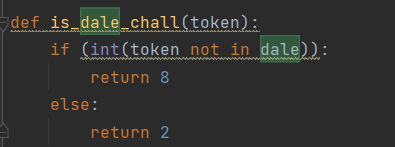
Am decis în a împărții implementarea în 4 etape secvențiale, prima etapă constând în implementearea unui feature, testarea acestui feature, verificarea manuală a setului de date în funcție de featureul realizat(opțional) și implementarea acestuia in algoritmul întreg.

Pentru algoritmul realizat am încercat două tipuri de clasificatori. Primul a fost KNN (K Nearest Neighbors) și Gaussian Naive Bayes.

# Prezentare si explicare featurerurilor folosite

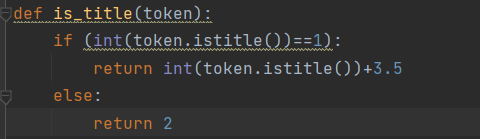
Parametrii aleși pentru featurerui au fost aleși în fucție de impactul pe care l-am considerat că îl are acel feature pentru a descoperi dacă un cuvânt este sau nu complex. Cu cât parametrul este mai mare cu atât cuvântul are șansa mai mare să fie complex. Ca puncte de reper am ales 2 ca șansă foarte mică să fie complex, 5 ca șansă medie și 8 ca șansă bună.

**is\_dale\_chall:**

****

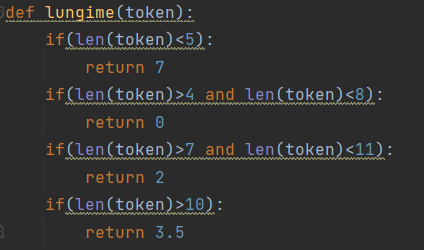
Funcția folosește librăria „dale\_chall” pentru a verifica dacă cuvântul analizat face sau nu parte din setul de cuvinte familiare. Astfel, cuvântul daca nu face parte din seria respectivă va primi ca parametru 2(șanse foarte mici sa fie complex) iar dacă nu este 8(șansă bună să fie complex)

**is\_title:**



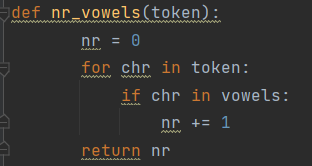
Funcția verifică dacă cuvântul primit începe sau nu cu majusculă. Pentru acest feature am ales să fac și o verificare a setului de date rezultând faptul că din numărul total de cuvinte complexe 750, 304 încep cu majusculă. Iar pentru cele 6912 cuvinte necomplexe, doar 1304 încep cu majusculă. Așadar, am comparat aceste date la nivel procentual și am ajuns la concluzia că tokenurile ce încep cu majusculă sunt mai preponderente în setul de cuvinte complexe decât în cel de necomplexe (șansă de aproximativ 50% pentru complex față de o șansă doar de aproximativ 20% pentru necomplex). De aceea am ales ca cuvântul să primească parametru 4.5(o șansă medie) în cazul în care începe cu majusculă și 2 în caz contrar.

**lungime:**



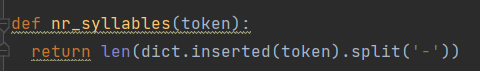
Funcția verifică lungimea totală a tokenului. Pentru acest feature am făcut iar o verificare a setului de date pentru a îi alege impactul. Așadar cuvintele în intervalul 11+ litere a avut o rată aproximativ de 9 la 7 pentru a fi necomplex (901 cuvinte necomplexe din 6912 vs 107 cuvinte complexe din 750). Cuvintele pentru o lungime cuprinsă între 8 și 10 litere rata a fost de aprox. 2.5 la 2.5 (2168 necomplex vs 238 complex), pentru intervalul 5 și 7 o rată de aprox. 2.5 la 4 (2956 necomplex vs 208 complex) și pentru cuvinte sub 5 litere, o rată de 9 la 3 (887 necomplex vs 197 complex). În funcție de aceste proporții am ales parametrii optimi.

**nr\_vowels:**



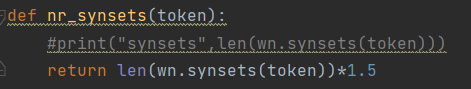
Funcția verifică numărul de vocale iar cu cât cuvântul conține mai multe vocale acesta are o șansă mai mare să fie complex.

**nr\_syllables:**



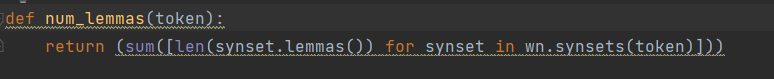
Funcția se folosește de librăria Pyphen pentru a crea dicționarul englez de cuvinte în engleză astfel acestea putând să verifice numărul de silabe ale tokenului. Așadar acesta are o șansă mai mare să fie complex cu creșterea numărului de silabe.

**nr\_synsets:**



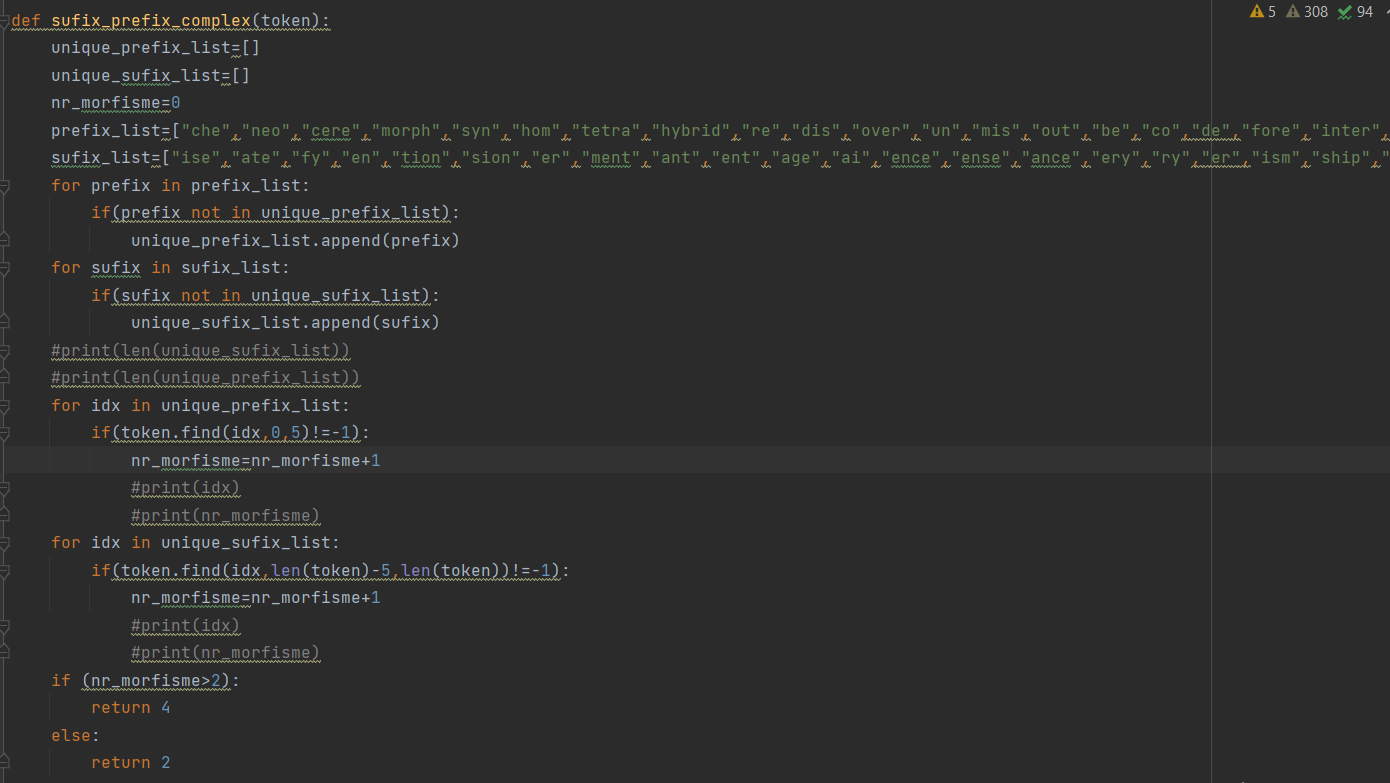
Funcția se folosește de librăria NLTK, Wordnet pentru a găsi grupurile de sinonime care au la bază același concept. Deoarece am considerat că numărul de grupări influențează preponderent complexitatea cuvântului am ales creșterea acestui parametru.

**num\_lemmas (neutilizată pentru cel mai eficient algoritm):**



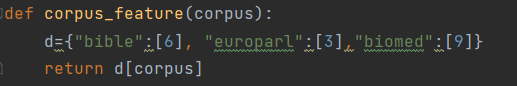
Funcția se folosește de librăria NLTK, Wordnet pentru a găsi grupurile de sinonime care au la bază același concept iar mai apoi pentru fiecare dintre acestea să caute forma canonică și sinonimele acesteia. Am decis să renunț la această funcție deoarece nu aducea beneficiu codului fiind puțin similară cu funcția precedentă.

**sufix\_prefix\_complex (neutilizată pentru cel mai eficient algoritm):**



Funcția creează 2 liste de sufixe si prefixe bazate pe cele mai comune sufixe si prefixe din limba engleză și mai apoi verifică dacă tokenul conține aceste morfisme. Este considerat un cuvânt complex dacă conține un număr mai mare de morfisme (2+). Așadar acest feature ar trebui sa verifice numărul de morfisme din token. În urma verificării și testării acestui feature am constatat că implementarea nu este una destul de eficientă pentru a imbunătății algoritmul.

**corpus\_feature:**



Funcția verifică din ce corpus face parte cuvântul, deoarece în urma analizării datelor de antrenare, am observat ca tokenul care face parte dintr-un anumit corpus are șanse mai mari să fie complex. În urma analizării cuvintele din categoria „Bible” din 2574 de cuvinte, 236 sunt complexe, din categoria „Biomed” din 2576, 392 sunt complexe iar din categoria „Europarl” din 2512, 122 sunt complexe. Așadar șansele ca și cuvintele din categoria „Biomed” au șanse mai mari să fie complexe față de cele din „Europarl”, iar parametrii au fost aleși după acest principiu.

**dale\_chall\_prop:**



Funcția verifică complexitatea pe care o are o propoziție folosindu-se de „Dale-Chall readability formula”, complexitatea propoziție în funcție de numărul de cuvinte complexe, adaptată pentru setul nostru de date. Mai întâi am realizat o preprocesare a propoziției pentru a elimina spațiile și semnele de punctuație apoi am folosit formula pe datele acestea. Deoarece formula este pentru texte ce conțin de mai multe propoziții am încercat o adaptare a acesteia iar în urma testării am ajuns la varianta finală. Parametrii returnați au la bază scorul propriu-zis al formulei, 9+ ințeleasă ușor de o persoană din clasa a 13-a sau mai mult, de aceea am ales ca șansele ca un cuvânt sa fie complex sunt maxime la un scor de peste 11.

**gunning\_readablity (neutilizată pentru cel mai eficient algoritm):**



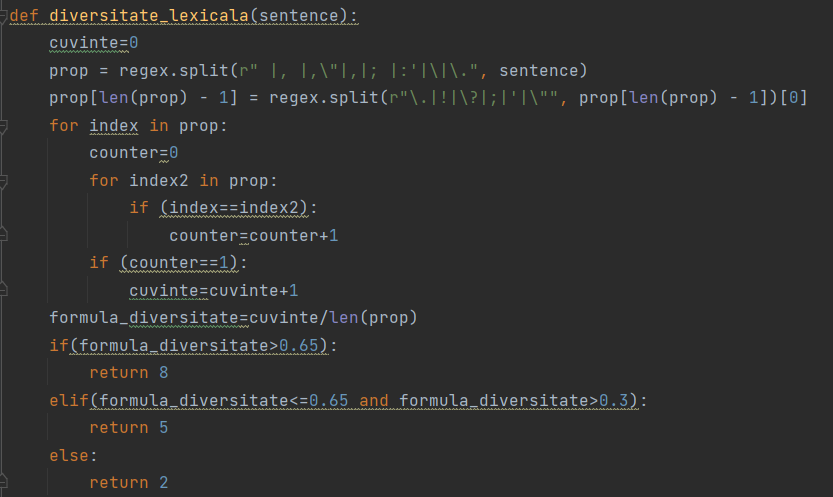
Funcția verifică complexitatea propoziție în funcție de complexitatea cuvintelor calculată pe baza numărului de silabe, un cuvânt complex fiind considerat daca conține cel puțin 3 silabe. Ca în metoda precedentă am realizat o preprocesare apoi folosindu-mă de „Gunning Readability formula” am calculat complexitatea propoziției.

**flesh\_readability (neutilizată pentru cel mai eficient algoritm):**



Funcția verifică complexitatea propoziție în funcție de numărul de silabe, fiind o abordare similară cu metoda precedentă am ales să o implementez deoarece ambele formule sunt folosite pentru texte ce conțin mai multe propoziții și cum setul de date primit conține doar texte cu o propoziție pot apărea erori, așadar am ales implementarea unei medii între cei 2 parametrii returnați pentru a crea un singur feature.

**diversitate\_lexicala (neutilizată pentru cel mai eficient algoritm):**



Funcția verifică complexitatea unei propoziții în funcție de diversitatea cuvintelor. Ca în cazurile precedente am realizat o preprocesare pentru eliminarea caracterelor fără utilitate, apoi în funcție de aparițiile cuvintelor am calculat formula de diversitate, iar pe baza unor standarde ale rezultatului am setat parametrii.

# 

# Clasificatori

În cadrul algoritmului am folosit 2 clasificatori, la început am folosit K Nearest Neighbors iar mai apoi pentru a îmbunătății rata de succes, Gaussian Naive Bayes.

Modelul celor mai apropiati vecini este un model de invatare supervizata având etichete care prezice eticheta unui exemplu de test ca fiind eticheta predominantă ale celor mai apropiate k exemple de antrenare. Modelul KNN nu invata in mod explicit un model anume, ci stocheaza in memorie toate datele de antrenare si face predictia pe baza acestora.

În cazul hiperparametrilor, am ales pentru aceștia valorile experimental în urma testelor realizate pe platforma de concurs. Am ales K-ul impar fiind vorba de o clasificare complex/necomplex (binară). Am început testarea k-ului cu 1 iar apoi crescut până la 10 și cu 20/30/40/50/100. În urma testelor am constat că pentru 5 am avut cele mai bune submisii având o eficiență de aproape 30% peste restul de k-uri.

Pentru KNN cel mai bun scor obținut 0.57072 pe 40% din setul de date, iar durata antrenării a fost în medie de 16.82 milisecunde, timpul fiind direct proporțional cu volumul de date de antrenare.

Am preferat să schimb clasificatorul, în Gaussian Naive Bayes. Aceasta are la bază teorema lui Bayes pentru date care au o distribuție normală, gaussian.   
Teorema lui Bayes este: , pentru cazul de față „c” va reprezenta tipul cuvântului, complex sau necomplex iar „x” va reprezenta token. Așadar, formula pentru cazul complex va deveni:

Iar reprezintă probabilitatea likelihood, adică probabilitatea pe care o avem să găsim cuvântul respectiv la setul de cuvinte complexe, reprezintă probabilitatea ca și un cuvânt să fie complex iar reprezintă probabilitatea pe care o are un cuvânt să fie exact cuvântul dat.

Legat de hiperparametrii am decis în a le păstra pe cele prestabilite.

Pentru Gaussian Naive Bayes, cel mai bun scor obținut a fost de 0.78989 pe 40% din setul de date, cu o durată de 2.44 milisecunde. În cazul testării cu „K Fold Cross Validation” unde K-ul este 10 pentru cele 10 combinații de atrenare și testare, eficiența algoritmului a fost de: 0.7274936601859678; 0.7507610789980732; 0.7342752583536065; 0.7315263240118732; 0.6976623376623377; 0.7199624196613499; 0.771742794539019; 0.7087177406551536; 0.7407309671460616; 0.7340379008746356. În funcție de această distribuire am realizat și matricea de confuzie pentru întregul set de date, aceasta avand forma:

[4504 2408 ]

[ 142 608 ]

Antrenarea în ambele cazuri a constat în crearea de vector caracteristic pentru fiecare cuvânt în parte folosindu-mă de setul de featureri prezentat mai inițial. Astfel am creat un vector ce reflectă structura și forma cuvântului, a propoziției din care face parte cât și a corpusului. Detalii pe care le-am considerat relevante în deducerea complexității tokenului.