**Student: Chiriac Nicu-Manuel**

**Predictie SII**

**-documentație-**

Cuprins

[1.Biblioteci,clase și funcții externe necesare 1](#_Toc124872416)

[2. Preprocesări ale textului 2](#_Toc124872417)

[3. Extragere de feature-uri 2](#_Toc124872418)

[4. Crearea de dicționar și asocierea valorilor obținute 2](#_Toc124872419)

[5. Antrenarea modelului și rezultate 2](#_Toc124872420)

# 1.Biblioteci,clase și funcții externe necesare

Pentru realizarea predicției au fost necesare următoarele biblioteci, după cum sunt prezentate în tabelul 1.1

|  |  |
| --- | --- |
| Tabel 1.1 - Biblioteci | |
| ***re*** | bibliotecă de expresii regulate pentru Python |
| ***nltk (Natural Language Toolkit)*** | Folosit pentru sarcinile de procesare a limbajului natural în Python |
| ***stopwords*** | este un corpus din biblioteca nltk care conține o listă de cuvinte utilizate în mod obișnuit în texte, care sunt adesea ignorate în sarcinile NLP |
| ***PorterStemmer*** | este o clasă din biblioteca nltk, care este folosită pentru stemming |
| ***panda*** | util pentru manipularea și analiza datelor |
| ***numpy*** | folosit pentru operații numerice în Python |
| ***TextBlob*** | bibliotecă pentru procesarea datelor dintr-un text, oferă suport în sarcini comune de procesare a limbajului natural, cum ar fi etichetarea unei părți din vorbire, analiza sentimentelor, clasificare, traducere și multe altele. |
| ***sklearn.ensemble.*** ***RandomForestClassifier*** | este o clasă pentru algoritmul Random Forest din biblioteca scikit-learn |
| ***sklearn.preprocessing.MinMaxScaler*** | este o clasă din biblioteca scikit-learn pentru scalarea datelor între o valoare minimă și maximă dată |
| ***word\_tokenize*** | utilizat pentru tokenizarea textului (împărțirea lui în cuvinte individuale) |
| ***FreqDist*** | Folosit pentru numărarea frecvenței cuvintelor |
| ***cmudict*** | corpus din biblioteca nltk care conține pronunții fonetice ale cuvintelor |
| ***textstat*** | bibliotecă Python pentru statistici de text |
| ***sklearn.model\_selection.GridSearchCV*** | ajuta la crearea unui *grid* pentru obținerea combinațiilor de hyperparametrii (cross-validation) |

# 2. Preprocesări ale textului

A fost creata funcția *preprocess\_text(text)* care ulterior a fost aplicată pe serul de date. Setul de date - fișierul texts.csv a fost atribuit unui dataFrame cu ajutorul bibliotecii *pandas*.

*Preprocess\_text()* realizează următoarele sarcini :

1. Conversia din majuscule în minuscule (lowerCase)
2. Transformarea textului in token-uri
3. Eliminarea stop-word-urilor
4. Lematizare

O imagine care conține masă

Descriere generată automat

Fig. - Texte preprocesate

# 3. Extragere de feature-uri

Mai departe a fost necesară crearea unor noi funcții care să extragă o serie de feature-uri care au fost considerate relevante. Acestea sunt:

1) *lungimea textului*

2) *complexitatea vocabularului* (calculat folosind formula *unique\_tokens / len(tokens) )*

*3) scor de lizibilitate* sau *scor de subiectivitate* (obținut utilizând biblioteca *TextBlob*)

4) *frecvența părților de vorbire*

5) *Flesch-Kincaid*, calculat cu formula:

score = 206.835 - 1.015 \* (num\_words / num\_sentences) - 84.6 \* (num\_syllables / num\_words)

S-a constatat că o mare parte din valorile calculate pentru părțile de vorbire aveau valori fie mult prea mari (eronate, întrucât nr. părților de vorbire depășea totalitatea cuvintelor), fie 0, astfel alegerea a fost să fie scoase din lista de *pos\_tags* . Au fost păstrate **JJ** – adjective, **VBP** – verb, sing. present, non-3d take , **RB** – adverbe.

# 4. Obținerea scorurilor, asocierea lor și crearea de dicționare

Mai departe au fost obținute scorurile feature-urile mai sus menționate, folosind funcția *apply()* din pandas care aplică o funcție custom fiecărui element *text* din *text\_df.* Funcția *lambda* este o funcție anonimă care ia drept parametru *text* și este aplicată tuturor elementelor de tip text din dataFrame.

Pentru a evita bias-ul generat de diferența de scară dintre feature-uri, toate datele au fost scalate folosind obiectul *scaler,* implicit clasa *MinMaxScaler.* Metoda *fit()* calculează minimul și maximul valorilor. *reshape(-1, 1)* remodelează un vector unidimensional într-unul bidimensional, întrucât scaler-ul așteaptă un vector bidimensional.

Acește feature-uri sunt adăugate data frame-ului, așa cum este prezentat în figura 2.

O imagine care conține masă

Descriere generată automat

Fig. 2 - texts\_df with features

Mai departe sunt citite și adăugate in data frame uri dataset-urile de train, validation și test. Este creată o variabilă de tip listă care conține toate feature-urile. Este creat un dicționar și toate valorile/scorurile sunt asociate conform id-urilor. S-a dat drop la coloanele de text, întrucât acestea nu sunt relevante modelului.

Suplimentar s-a făcut o verificare și s-a observat că în jur de 80 de înregistrări aveau 2 pe coloana ‚Result’, acestea fiind înlocuite cu 1.

# 5. Antrenarea modelului și rezultate

Au fost testate 2 modele : Logistic Regression si Random Forest. Ambele oferă rezultate bune în task-urile de clasificare.

Varianta finală a fost Random Forest, întrucât acesta a avut o acuratețe mai mare.

Până la obținerea scorului de acuratețe, a fost necesară antrenarea modelului. S-au stabilit intrările și ieșirile. X\_train (intrările) au fost atributele/feature-urile extrase, iar Y\_train coloana ‘Result’. În aceeași manieră s-au stabilit și pentru setul de validare și test.

Pentru a obține un scor mai bun, s-a utilizat GridSearchCV din sklearn, cu ajutorul căruia se creează un grid și se poate face hyperparameter tuning, implicit cross-validation (combinații de cei mai buni hiperparametrii). Gridul este format dintr-un număr de *n\_estimators* (număr de arbori de decizie, fiecare scoțând propria predicție) și *max\_depth* (adâncimea fiecărui arbore).

A fost instanțiat modelul și a fost creat GridSearchCV, care ia următorii parametrii : modelul, gridul de parametrii și ‘cv’ care reprezintă numărul de *folds* pentru cross-validation. Apoi modelul este antrenat, folosind *best\_model.fit(X\_train, y\_train),*  best\_model fiind *RandomForestClassier*, care ia drept argument *best\_params.*

Acuratețea obținută a fost de 0.63.

În ultimă instanță este creat un nou fișier csv denumit *final\_predictions,* fiind de fapt setul de test cu ultima coloana “Result” completată cu predicția modelului (0 sau1).