Documentație proiect clasificare text

I. Preambul

Prezentul proiect se constituie sub forma unui program în limbajul Python, dezvoltat în cadrul IDE-ului de la Jetbrains, Pycharm, și utilizează principii ale învățării autonome predate la cursul de Inteligență Artificială. Scopul acestei implementări este acela de a putea prelucra fișiere care conțin propoziții scrise în trei limbi extraterestre, cu ajutorul algoritmilor de machine learning, astfel încât să se poată prezice pentru orice propoziție limbajul utilizat.

II. Preluarea datelor

Fișierele care conțin datele de input sunt cele utilizate în general în procesul de învățare automată:

- train_samples.txt și train_labels.txt - conțin datele (propozițiile și etichetele corespunzătoare) ce vor fi utilizate pentru antrenarea modelului;
- validation_samples.txt și validation_labels.txt - conțin datele (propozițiile și etichetele corespunzătoare) ce vor fi utilizate pentru verificarea eficienței pe care o atinge modelul;
- test_samples.txt - propozițiile pe baza cărora se vor face predicțiile pentru concursul Kaggle.

Datele de intrare nu au o structură uniformă în cadrul fișierelor de intrare (conțin identificatori la început de rând, lungimea propozițiilor nu este uniformă), astfel că citirea lor nu s-a putut realiza cu o funcție din numpy (spre exemplu: np.loadtxt). În schimb, pentru preluarea inputului am folosit o procesare manuală a datelor: citirea rând cu rând a propozițiilor din fișier, eliminarea elementelor neimportante (identificatorii, caracterele de formatare - /n) și stocarea propozițiilor în liste corespunzătoare fiecărui document.

Fișierele de ieșire sunt reprezentate de fișierul CSV (RESULT.csv) cu predicțiile pe baza datelor de test și de fișierele text utilizate de mine pentru înregistrarea datelor pe parcursul testelor cu modelele de antrenare.

III. Standardizarea inițială a datelor:

Pornind de la setul de date citit, se efectuează o standardizare inițială a acestuia cu ajutorul metodelor CountVectorizer(), având parametrii default, fit_transform() și

transform() din cadrul bibliotecii sklearn. Scopul pentru care se realizează această formatare a datelor este pentru a se face trecerea de la datele inițiale de tip string din propoziții către datele de tip int ce pot fi prelucrate de către algoritmii de ML.

Astfel, se va construi pe baza propozițiilor inițiale un vocabular ce va fi folosit pentru realizarea de vectori caracteristici pentru fiecare dintre enunțurile inițiale (se contorizează apariția fiecărui cuvânt din vocabular). Ulterior, aceştia vor fi cuprinşi într-o nouă matrice, echivalentă cu textul inițial, dar pe baza căreia se pot realiza operații statistice: normalizări, medii, deviații standard, necesare efectuării de predicții.

IV. Alegerea modelelor pentru antrenare:

Pentru a determina modelul care se pretează cel mai bine problemei de identificare a limbii utilizate într-un text, am trecut prin mai multe tipuri de clasificatori: cei probabilistici - Naive Bayes cu derivate acestuia (Multinomial, Complement și Gaussian) , cei liniari - Stochastic Gradient Descent (SGD), Support vector machines (SVM) și Perceptron și prin cei cu rețele neuronale - Multi-layer Perceptron (MLP). Rezultatele au fost următoarele:

```
   Clasificator      Scor
1. MultinomialNB() : 0.6854
2. ComplementNB() : 0.6774
3. MLPClassifier(early_stopping=True, verbose=True) : 0.6772
4. SGDClassifier() : 0.6346
5. SGDClassifier(eta0=1, learning_rate='constant', loss='perceptron', penalty=None) : 0.6318
6. SVM() : 0.61357
7. GaussianNB() : 0.5546
```

Se poate observa că modelele de antrenare cu Naive Bayes (Multinomial și Complement) și cel cu rețele de neuroni (MLPClassifier) au reușit să obțină cele mai bune scoruri în cazul unui input cu o standardizare a datelor cu setările default. Acestea vor fi folosite în continuare în proiect.

V. Optimizarea procesării datelor:

Pentru creșterea performanței predicției algoritmilor de învățare, setul de date va trebui îmbunătățit, astfel că se va renunța la valorile implicite ale parametrilor și se vor oferi unele noi. Argumentele modificate de mine în cadrul proiectului au fost:

- ➢ pentru CountVectorizer():
  - **ngram_range**: preia subsecvențe de text și le compară;

- **max_df**: elimină cuvintele cele mai întâlnite în vocabular și care nu oferă informație relevantă clasificării;
- **analyzer**: `char_wb` - asupra căror elemente se va realiza analiza, respectiv ngram asupra caracterelor numai din interiorul cuvintelor;
- **strip_accents**: `unicode` - eliminarea caracterelor și normalizarea acestora.

➢ pentru clasificatori: hiperparametrul alfa - nu aduce nicio îmbunătățire modelelor testate de mine - rămâne cu valoarea implicită 1.

1. MLPClassifier

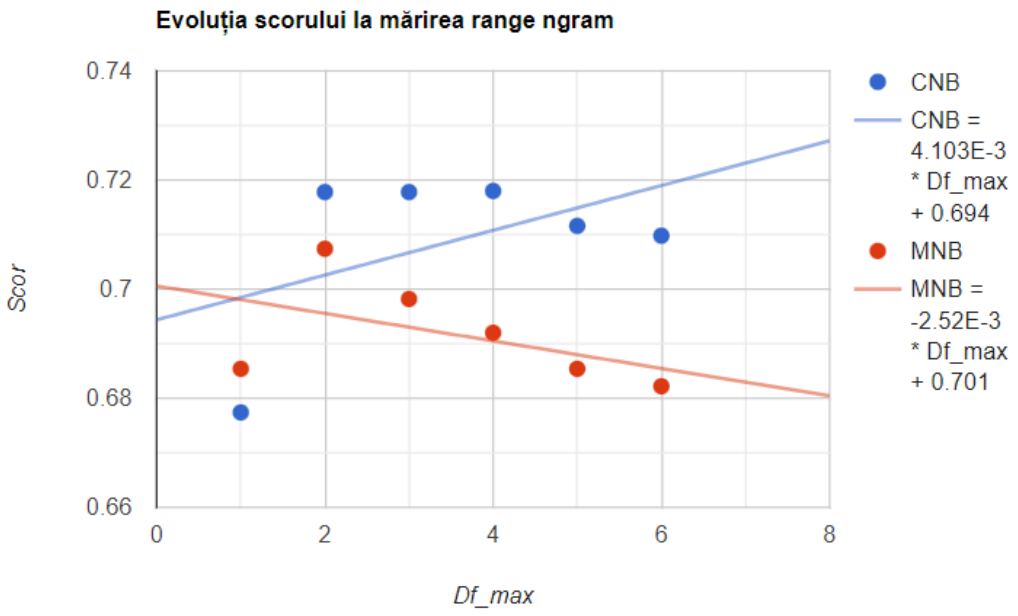
Pentru clasificatorul cu rețea de neuroni nu s-au putut aduce îmbunătățiri semnificative, întrucât atât timpul de antrenare a crescut exponențial odată cu introducerea ngramelor (mai mult de o oră și jumătate și modelul nu era finalizat), cât și cererea pentru resurse (probleme cu spațiul alocat programului, care ajungea ca cerere până la 16 GB).

2. MultinomialNB vs ComplementNB

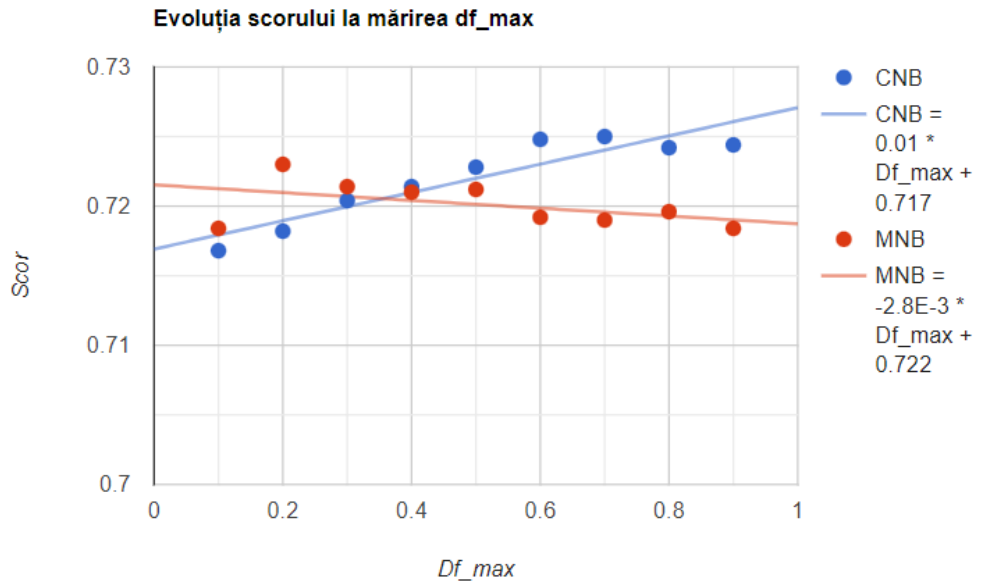
Cele două tipuri de clasificatori au la bază principiul probabilistic Naive Bayes, însă se raportează diferit la distribuția datelor. MultinomialNB pleacă de la prezumția că propozițiile sunt distribuite uniform, deci că există o cantitate uniformă de enunțuri din fiecare limbă, în timp ce ComplementNB ia în calcul o eventuală asimetrie în cadrul cantităților de date din fiecare limbă.

În urma testelor de optimizare rezultatele au fost similare între cele două modele, acuratețea fiind puțin mai mare la CNB (0.725 - best case) față de MNB (0.723 - best case).

Valorile pentru **ngram_range** și **max_df** fluctuează ca performanță de la un model la altul. Astfel, CNB are o creștere a scorului de îndată ce se mărește orizontul ngram, în timp ce la MNB se poate observa o stabilizare în timp.



În ceea ce privește creșterea scorului în urma schimbării valorii df_max (la ngram_range(1,5) pentru ambele modele), se poate observa că există o distribuție apropiată de cea Gaussiană



Utilizarea împreună a **analyzer**: `char_wb` și **strip_accents**: `unicode`duce la o creștere a performanței modelului, indiferent de tipul modelului testat.

caz CNB	scor
fără param	0.7178
cu param	0.725

caz MNB	scor
fără param	0.708
cu param	0.723

Utilizarea unei formatări a datelor de tip TFIDF (”Term Frequency – Inverse Document Frequency”) nu a îmbunătățit scorul. Aceasta a constat în calcularea unui scor pentru fiecare cuvânt unde frecvența generală era corelată cu frecvența absolută în cadrul tuturor textelor.

```
tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
```

Accuracy score și confusion matrix pentru cazul MultinomialNB cu `CountVectorizer(ngram_range=(1, 5), max_df=0.2, analyzer='char_wb', strip_accents='unicode')` :

```
0.723
[[1426  371  203]
 [ 326 1003  171]
 [ 233   81 1186]]
```

Accuracy score și confusion matrix pentru cazul ComplementNB cu `CountVectorizer(ngram_range=(1, 5), max_df=0.7, analyzer='char_wb',`

`strip_accents='unicode')` , reprezentând best case obținut de mine în cadrul acestui proiect:

```
0.725
[[1383  331  286]
 [ 292  980  228]
 [ 187   51 1262]]
```

VI. Concluzii

În cazul testelor efectuate de mine, algoritmii de învățare de tip Native Bayes au avut cele mai bune performanțe, obținând pentru ComplementNB scorul de 0.725 pe datele de validare și 0.71394 în cadrul concursului susținut pe platforma Kaggel.