UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI ELECTRONICĂ

DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

PROIECT DE DIPLOMĂ

Andrei - Ciprian Amzuloiu

COORDONATOR ȘTIINȚIFIC

Conf. dr. ing. Marian - Cristian Mihăescu

IULIE 2021

CRAIOVA

UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ŞI ELECTRONICĂ

DEPARTAMENTUL DE CALCULATOARE ŞI TEHNOLOGIA INFORMAȚIEI

Detectarea plagiatului în text

Andrei – Ciprian Amzuloiu

COORDONATOR ȘTIINȚIFIC

Conf. dr. ing. Marian - Cristian Mihăescu

IULIE 2021

CRAIOVA

„Toate sunt trecătoare. Chinuri, suferințe, vărsări de sânge, molimă, foamete. Totul va pieri, dar stelele de pe cer vor dăinui și atunci când jos, pe pământ, nu va mai rămâne nici măcar umbra noastră sau a înfăptuirilor noastre. Nu se află nimeni pe fața pământului care să n-o știe. Și atunci de ce nu vrem să ne îndreptăm privirea spre stele? De ce?”

Mihail Bulgakov, Garda Alba

DECLARAȚIE DE ORIGINALITATE

Subsemnatul *ANDREI – CIPRIAN AMZULOIU*, student la specializarea *CALCULATOARE ROMÂNĂ* din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul *Detectarea plagiatului în text*,
- coordonată de *Conf. dr. ing. MARIAN – CRISTIAN MIHAESCU*,
- prezentată în sesiunea *IULIE 2021*

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acţiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
- însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data, Semnătura candidatului,

--.--.--

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare şi Tehnologia Informaţiei

Aprobat la data de
……
Şef de departament,
Prof. dr. ing.
Marius BREZOVAN

PROIECTUL DE DIPLOMĂ

Numele şi prenumele studentului/-ei:	*Amzuloiu Andrei - Ciprian*
Enunţul temei:	*Detectarea plagiatului în text.*
Datele de pornire:	*Realizarea unei implementări pentru detectarea plagiatului în text şi evaluarea rezultatelor pe un set de documente în care s-a inserat plagiat artificial de diferite complexităţi.*
Conţinutul proiectului:	
Material grafic obligatoriu:	1. Documentaţia proiectului 2. Codul sursă 3. Prezentarea Power Point
Consultaţii:	*De două ori pe lună.*
Conducătorul ştiinţific (titlul, nume şi prenume, semnătura):	*Conf. dr. ing. MARIAN – CRISTIAN MIHAESCU*
Data eliberării temei:	
Termenul estimat de predare a proiectului:	
Data predării proiectului de către student şi semnătura acestuia:	--.--.-- 

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Calculatoare şi Tehnologia Informației

REFERATUL CONDUCĂTORULUI ŞTIINȚIFIC

Numele și prenumele candidatului/-ei:
Specializarea: Calculatoare
Titlul proiectului: Detectarea plagiatului în text

Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):
În facultate X
În producție ☐
În cercetare ☐
Altă locație:

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Tipul proiectului		Cercetare ☐	Proiectare ☐	Realizare practică ☐	Altul
Aparatul matematic utilizat		Simplu ☐	Mediu ☐	Complex ☐	Absent ☐
Utilitate		Contract de cercetare ☐	Cercetare internă ☐	Utilare ☐	Altul
Redactarea lucrării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Partea grafică, desene		Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
Realizarea practică	Contribuția autorului	Insuficientă ☐	Satisfăcătoare ☐	Mare ☐	Foarte mare ☐
	Complexitatea temei	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐
	Analiza cerințelor	Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
	Arhitectura	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐

	Întocmirea specificațiilor funcționale	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Implementarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Testarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Funcționarea	Da ☐	Parțială ☐	Nu ☐	
Rezultate experimentale		Experiment propriu ☐		Preluare din bibliografie ☐	
Bibliografie		Cărți	Reviste	Articole	Referințe web
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI	RESPINGEREA PROIECTULUI
☐	☐

Data, Semnătura conducătorului științific,

REZUMATUL PROIECTULUI

Proiectul are ca scop realizarea unui pipeline de algoritmi / metode prin intermediul căruia, dându-se un set de documente suspicioase și de documente sursă, să se poată găsi pasajele din documentele suspicioase care au fost plagiate și pasajele de text corespunzătoare din cadrul documentelor sursă.

Corpusul folosit (care conține documentele) menționate este *PAN-PC-09*, folosit în cadrul competiției *1st International Competition on Plagiarism Detection.* Acesta conține documente de diverse dimensiuni (de la dimensiuni mici, o pagina, pana la dimensiuni mari, 100 – 1000 de pagini), cu un raţio de 50% documente suspicioase, 50% documente sursa. Secțiunile plagiate sunt realizate în mod artificial, fiind preluate pasaje din documentele sursă în care s-au aplicat (la un anumit nivel) diverse modificări (*traduceri*, *Random Text Operation*, *Semantic Word Variation* și *preserving Word Shuffling*).

De asemenea, metodele de validare ale detecției au fost implementate orientat pe sistem (macro-averaged), modalitate descrisă în cadrul concursului.

Procesarea limbajului natural este una dintre componentele cheie din domeniul Inteligenței Artificiale, care oferă abilitatea de a face un calculator să recunoască limbajul uman. Aceasta tehnica ajută calculatoarele să înțeleagă și să extragă diferite pasaje dintr-un set de date de tip text prin aplicarea diferitelor tehnici cum ar fi, similaritatea textului, regăsirea informației, clasificarea documentelor, extragerea entităților și, într-un final, clustering. Detecția plagiatului reprezintă o temă în care tehnicile de procesare ale limbajului natural pot aduce multe contribuții, dat fiind că un text plagiat poate fi supus unei serii largi de modificări.

Pentru realizarea pipeline-ului menționat, m-am folosit de următorii pași:

1. Selectarea documentelor candidat pe baza rezultatelor descoperite prin metoda *Encoplot*.
2. Preprocesarea segmentelor de text determinate din documentele candidat.
3. Calcularea valorilor de tip *embeddings* pentru segmentele candidat preprocesare folosind un model de tip *transformer*.
4. Compararea segmentelor preprocesare folosind similitudinea cosinus și extragerea unei serii de segmente candidat.

5. Determinarea secțiunilor plagiate folosind DBSCAN (Density-based spatial clustering of applications with noise) asupra segmentelor candidat. Un cluster va fi corespunzător unei secțiuni de text plagiate.

6. Validarea rezultatului.

Soluția a fost realizată folosindu-mă de limbajul de programare Python, care reprezintă „lingua franca” pentru domeniul procesării de limbaj natural, cât și pentru alte domenii legate de Inteligența artificială. Platforma pe care s-a făcut implementarea este Google Colab, datorită necesității de putere de procesare mare.

Termenii cheie: Transformer, BERT, Machine learning, Python, Word Embeddings, Encoplot, N-gram, DBSCAN, Similaritate Cosinus

MULȚUMIRI

Mulțumesc pe această cale coordonatorului meu științific, domnul Conf. Dr. Ing. Marian Cristian Mihăescu, cât și domnului Conf. Dr. Ing. Traian Rebedea, pentru tot sprijinul informațional oferit pentru realizarea acestui proiect.

De asemenea, aș dori să mulțumesc familiei pentru tot sprijinul moral și emoțional oferit, atât pentru acest proiect, cât și de-a lungul anilor de studiu.

CUPRINSUL

LISTA FIGURILOR

LISTA TABELELOR

1 INTRODUCERE

1.1 Scopul

Scopul acestui proiect este acela de a realiza o soluție de a detecta într-un mod optim pasajele plagiate, cât și corespondentele lor din documentul sursă, folosind algoritmi și metode specifice de *procesare de limbaj natural* și *învățare automată*. Acest pipeline poate fi ulterior implementat în diferite aplicații de detectare a plagiatului (pagini web, aplicații mobile, aplicații desktop, etc.).

Setul de date folosit conține, de asemenea, câte un document XML pentru fiecare document, în care se regăsesc caracteristicile documentului (limba în care a fost scris, sursa din care a fost extras, secțiunile plagiate împreuna cu modul în care au fost realizate, poziția de start, dimensiunea și corespondentul din documentul sursă). Pe baza acestor informații, am putut implementa o metoda de a calcula un scor, prin intermediul căreia, pe perioada de dezvoltare am putut îmbunătăți constant soluția.

1.2 Motivația

Motivația vine atât din dorința proprie de a aprofunda domeniul de procesare de limbaj natural, cât și din dezvoltarea și rezultatele impresionante arătate de întreg domeniul învățării automate în ultimii ani.

2 INFORMAȚII DESPRE DOMENIU ȘI TEHNOLOGIILE FOLOSITE

Pentru a putea oferi un livrabil în cadrul proiectului de diplomă, diferite domenii, concepte, metode și limbaje de programare au fost studiate pe parcursul dezvoltării proiectului.

2.1 Învățarea automată

Învățarea automată este o particularizare a inteligenței artificiale (*AI – Artificial Intelligence*) care oferă sistemelor capacitatea de a „învăța” singure și de a se îmbunătăți din experiența acumulată fără a fi programate în mod explicit. Învățarea automată se concentrează pe dezvoltarea de modele care pot accesa datele oferite și le pot folosi pentru a se auto-îmbunătății.

Procesul de învățare începe cu observații sau date, cum ar fi exemple, experiență directă sau instrucțiuni, pentru a căuta modele în date și a lua decizii mai bune în viitor pe baza exemplelor pe care le oferim. Scopul principal este de a permite computerelor să învețe automat fără intervenția sau asistența umană și să ajusteze acțiunile în consecință.

Din perspectiva algoritmilor clasici de învățare automată, textul este considerat o succesiune de cuvinte cheie. Cu toate acestea, există abordări bazate pe analiza semantică care imită capacitatea umană de a înțelege semnificația unui text.

Algoritmii de învățare automată sunt adesea clasificați ca supravegheați sau nesupravegheați (supervised learning și unsupervised learning):

- Algoritmii de învățare automată *supravegheați* pot aplica ceea ce s-a învățat în trecut noilor date folosind exemple etichetate pentru a prezice evenimente viitoare. Pornind de la analiza unui set de date de antrenament cunoscut, algoritmul de învățare automată construiește un model pentru a face predicții despre valorile de ieșire. Sistemul este capabil să ofere rezultate relevante pentru orice intrare nouă după o pregătire suficientă. Algoritmul de învățare automată poate compara, de asemenea, ieșirea cu ieșirea corectă, și poate găsi erori pentru a modifica modelul în consecință.

- Algoritmii de învățare automată *nesupravegheați*, în contrast cu cei supravegheați, sunt utilizați atunci când informațiile utilizate pentru instruire nu sunt nici clasificate, nici etichetate. Învățarea fără supraveghere studiază modul în care sistemele pot deduce un model pentru a descrie o structură ascunsă din datele neetichetate. Sistemul nu e în căutarea unui rezultat corect, dar explorează datele și poate deduce rezultate pe baza tiparelor comune.

- Algoritmii de învățare automată *semi-supravegheați* se situează undeva între învățarea supravegheată și cea nesupravegheată, deoarece utilizează atât date etichetate cât și neetichetate pentru instruire - de obicei o cantitate mică de date etichetate și o cantitate mare de date neetichetate. Sistemele care utilizează această metodă sunt capabile să îmbunătățească considerabil acuratețea învățării. De obicei, învățarea semi-supravegheată este aleasă atunci când datele etichetate achiziționate necesită resurse calificate și relevante pentru a o instrui / învăța din ea. În caz contrar, achiziționarea de date fără etichetă nu necesită, în general, resurse suplimentare.

- Algoritmii de învățare automată de *întărire* (*reinforcement learning*) este o metodă de învățare care interacționează cu mediul prin producerea de acțiuni și descoperă erori sau recompense. Căutarea de încercări și erori și recompensa întârziată sunt cele mai relevante caracteristici ale învățării prin întărire. Această metodă permite mașinilor și agenților software să determine automat comportamentul ideal într-un context specific pentru a maximiza performanța acestuia. Este nevoie de feedback simplu pentru recompensă pentru ca agentul să afle care acțiune este cea mai bună. Acesta este cunoscut sub numele de semnal de întărire.

Învățarea automată permite analiza cantităților masive de date. Deși, în general, oferă rezultate mai rapide și mai precise pentru a identifica oportunități profitabile sau riscuri periculoase, poate necesita, de asemenea, timp și resurse suplimentare pentru a-l instrui în mod corespunzător. Combinarea învățării automate cu AI și tehnologiile cognitive o poate face și mai eficientă în procesarea unor volume mari de informații.

2.2 Procesarea limbajului natural

Prelucrarea limbajului natural (*NLP – Natural Language Processing*) este un domeniu la intersecția științei calculatoarelor, inteligenței artificiale și a lingvisticii. Se referă la construirea sistemelor care pot procesa și să înțeleagă limbajul uman. De la înființarea sa în anii 1950 și până foarte recent, procesarea limbajului natural a fost în primul rând un domeniu academic și al laboratoarelor de cercetare, necesitând o lungă educație formală și formare profesională. Descoperirile din ultimul deceniu au dus la folosirea tot mai mare a procesării de limbaj natural într-o serie de domenii diverse, cum ar fi comerțul, asistență medicală, finanțe, drept, marketing, resurse umane și multe altele. Din motivele dezvoltării recente a sistemelor bazate pe procesarea limbajului natural se încadrează următoarele:

- Instrumentele, tehnicile și API-urile bazate pe procesarea limbajului natural sunt disponibile pe scară largă și ușor de utilizat, în momentul de față fiind omniprezente în industrie. Nu a existat niciodată un moment mai bun pentru a crea rapid soluții bazate pe procesare de limbaj natural.

- Dezvoltarea unor abordări mai interpretabile și generalizate a îmbunătățit performanța de bază pentru probleme de procesare de limbaj natural mai complexe, cum ar fi simulări de conversație și răspuns la întrebări, care nu erau practic fezabile înainte.

- Din ce în ce mai multe companii, inclusiv Google, Microsoft și Amazon, investesc în produse de consum mai interactive, unde se folosește limbajul natural ca mijloc primar de comunicare.

- Apariția a din ce în ce mai multe seturilor de date open source, împreună cu standarde referitoare la acestea, a acționat ca un catalizator în această revoluție.

- Viabilitatea procesării de limbaj natural a depășit limba engleză sau alte limbi majore. Există seturi de date și modele specifice create și pentru limbile mai puțin digitalizate. Un rezultat al evoluției și lărgirii viabilității limbajului natural este reprezentat de aplicațiile de traducere (precum Google translate), care în momentul de față reușesc să ofere o traducere aproape perfectă pentru diverse limbi.

Există o serie de sarcini fundamentale care apar frecvent în diferite proiecte de procesare de limbaj natural. Datorită naturii lor repetitive și fundamentale, aceste sarcini au fost studiate pe larg. Pe scurt, aceste sarcini sunt următoarele:

- *Modelarea limbajului* - Această sarcină constă în a prezice care va fi următorul cuvânt dintr-o propoziție pe baza istoricului cuvintelor anterioare. Scopul sarcinii e de a „învățata” sistemul

de a determina probabilitatea ca un cuvânt sau a unei serii de cuvinte de a apărea într-un context. Modelarea limbajului este utilă pentru construirea de soluții pentru o mare varietate de probleme, cum ar fi recunoașterea vorbirii, recunoașterea optică a caracterelor, recunoașterea scrisului de mână, traducerea automată și corectarea ortografiei.

- *Clasificarea textului* - Această sarcină constă în a împărții o serie de texte într-un set cunoscut de categorii pe baza conținutului acestora. Clasificarea textului este de departe cea mai populară sarcină din domeniul procesării de limbaj natural și este utilizată într-o varietate de probleme, de la identificarea spamului prin e-mail până la analiza sentimentelor.

- *Extragerea informației* - După cum indică numele, această sarcină constă în a extrage informații relevante din text, cum ar fi evenimentele din calendar din e-mailuri sau numele persoanelor menționate într-o postare pe social media.

- *Regăsirea informației* - Această sarcină constă în a găsi documente relevante pentru o interogare a utilizatorului dintr-o colecție mare. Aplicațiile precum Google Search sunt cazuri de utilizare bine cunoscute de recuperare a informațiilor.

- *Agenți conversaționali* - Această sarcină constă în a construi sisteme de dialog care să poată conversa în limbi umane. Alexa, Siri și alți asistenți digitali sunt câteva exemple de aplicații care se folosesc de această sarcină.

- *Rezumarea textului* - Această sarcină își propune să creeze rezumate scurte ale documentelor mai lungi, păstrând în același timp conținutul de bază și păstrând sensul general al textului.

- *Răspunsul întrebărilor* - Această sarcină constă în a construi un sistem care poate răspunde automat la întrebările puse în limbaj natural.

- *Traducere automată* - Această sarcină constă în a converti o bucată de text dintr-o limbă în alta. Instrumentele precum Google Translate sunt aplicații obișnuite ale acestei sarcini.

- *Modelarea subiectelor* - Această sarcină constă în a descoperi structura actuală a unei mari colecții de documente. Modelarea subiectelor este un instrument comun de extragere a textului și este utilizată într-o gamă largă de domenii, de la literatură la bioinformatică.

Tehnicile de învățare automată sunt aplicate datelor textuale, așa cum sunt utilizate pe alte forme de date, cum ar fi imaginile, sunetele și alte tipuri de date. Tehnicile de învățare automată supravegheate, cum ar fi metodele de clasificare și regresie, sunt utilizate intens pentru diverse sarcini de procesare de limbaj natural. De exemplu, o sarcină de clasificare pentru procesarea de limbaj natural ar fi clasificarea articolelor de știri într-un set de subiecte de știri precum sportiv sau politic. Pe de altă parte, tehnicile de regresie, care oferă o predicție numerică, pot fi utilizate pentru a estima prețul unei

acțiuni pe baza procesării discuției pe social media despre acea acțiune. În mod similar, algoritmi de clustering nesupravegheați pot fi folosiți pentru a asocia documente text. Orice abordare de învățare automată pentru procesarea de limbaj natural, supravegheată sau nesupravegheată, poate fi descrisă ca fiind formată din trei paşi comuni: extragerea caracteristicilor din text, utilizarea reprezentării caracteristicilor pentru a învăța un model și evaluarea și îmbunătățirea modelului.

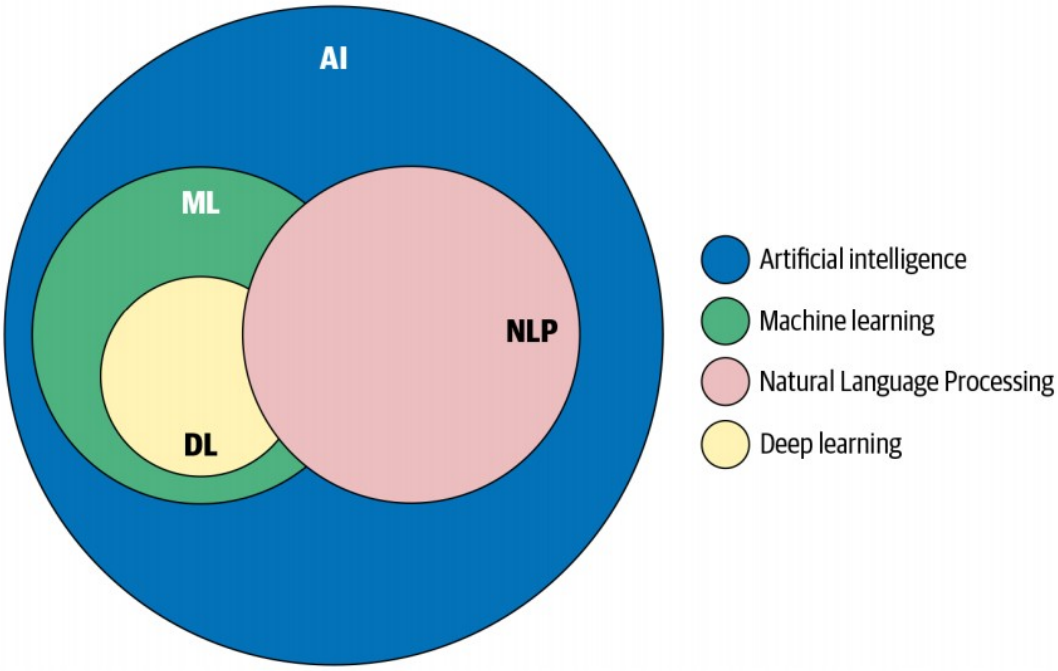


Figura 1: Poziția procesării limbajului natural faţă de AI, învâţarea automată și învăţarea profundă

2.3 METRICI DE EVALUARE

Ideea de a construi modele de învățare automată funcționează pe un principiu de feedback constructiv. Construind un model, se obține feedback din valorile prezise, astfel pașii următori constau în îmbunătățiri până când se obține o precizie dorită. Metricele de evaluare explică performanța unui model. Un aspect important al metricilor de evaluare este capacitatea lor de a discrimina rezultatele modelului.

În practică, luăm în considerare diferite tipuri de metrici de evaluare pentru a evalua modelele. Alegerea metricei depinde complet de tipul de model și de planul de implementare al modelului.

O matrice de confuzie este un tabel care este adesea utilizat pentru a descrie performanța unui model de clasificare (sau „clasificator”).

Matricea de confuzie are forma următoare:

		Predicted class	
		Positive	Negative
Actual class	Positive	TP	FP
	Negative	FN	TN

Tabelul 1: Forma unei matrici de confuzie

Astfel, în matricea de confuzie se vor regăsi următoarele tipuri de valori:

- *TP (True Positive)* este un rezultat în care modelul prezice corect o valoare ca fiind *„adevărată”*, aceasta fiind *„adevărată”* și în cadrul caracterisiticilor setului de date.
- *FP (False Positive)* este un rezultat în care modelul prezice greșit o valoare ca fiind *„adevărată”*, aceasta fiind de fapt *„falsă”* în cadrul caracterisiticilor setului de date.
- *FN (False Negative)* este un rezultat în care modelul prezice greșit o valoare ca fiind *„falsă”*, aceasta fiind *„adevărată”* în cadrul caracterisiticilor setului de date.
- *TN (True Negative)* este un rezultat în care modelul prezice corect o valoare ca fiind *„falsă”*, aceasta fiind *„falsă”* și în cadrul caracterisiticilor setului de date.

Considerând *TP ca* fiind numărul total de predicții corecte de tip „adevărat”, *TN* ca fiind numărul total de predicții corecte de tip „fals”, *FP* ca fiind numărul total de predicții greșite de tip „adevărat”, iar *FN* ca fiind numărul total de predicții greșite de tip „fals”, se pot defini următoarele formule pentru a defini precizia și reapelul:

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN}$$

Astfel, precizia reprezintă procentajul valorilor prezise corect ca fiind „adevărate”. În schimb, reapelul arată procentajul de valori prezise corect ca fiind „adevărate” din întreg setul de valori „adevărate”.

În imaginea următoare se poate observa într-un mod vizual rapoartele descrise de formulele menționate:

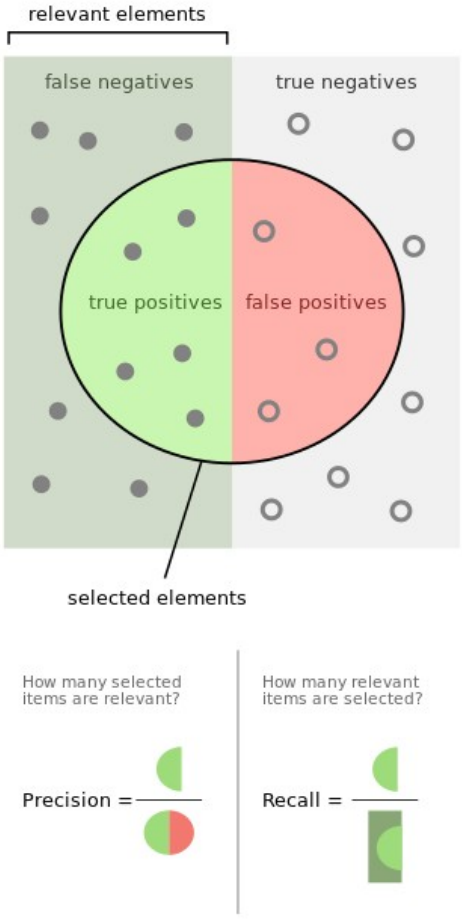
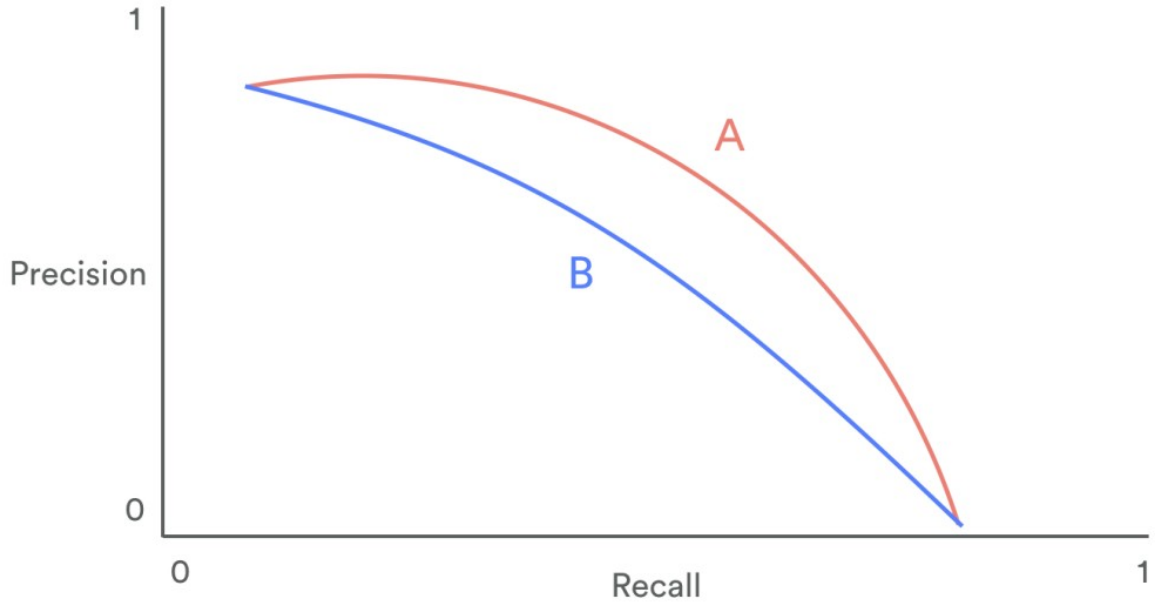


Figura 2: Reprezentare vizuală pentru precizie și reapel

Valorile preciziei și reapelului pot fi descrise de graficul următor:



În grafic se poate observa că de obicei, cu cât valoarea preciziei este mai mare, cu atât valoarea reapelului este mai mică. Acest fapt se aplică şi în sens invers, cu cât valoarea reapelului este mai mare, valoarea preciziei este mai mică. Astfel, limitarea metricilor de evaluare la o singură mărime din aceste două mărimi poate reprezenta o greşeală. De exemplu, dacă s-ar evalua modelul doar pe baza preciziei, ar fi suficient ca dintr-un set de date de dimensiuni mari să se facă o singură predicție true positive, iar precizia ar avea valori maxime (chiar dacă valoarea false negative este mare). În acelaşi mod, dacă s-ar evalua modelul doar pe baza reapelului, ar fi suficient ca toate predicțiile să fie adevărate (indiferent că este vorba de true positive sau false positve), iar reapelul ar lua valori maxime.

Scorul F_1 este o mărime care rezolvă problemele menționate anterior, acesta ținând cont atât de precizie, cât şi de reapel. Scorul F1 reprezintă media armonică dintre precizie şi reapel, şi este descris de formula următoare:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

2.4 Python

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să scrie cod într-o manieră mai clară și mai concisă decât în alte limbaje de programare. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software orientat pe obiect, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu garbage collector. Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

Implementarea de referință a Python este scrisă în C și poartă deci numele de CPython. Această implementare este open source și este administrată de fundația Python Software Foundation.

Printre avantajele lui Python se numără:

- *Ușor de învățat* - Python are puține cuvinte cheie, o structură simplă și o sintaxă clar definită.
- *Lizibil* - structura simplă și sintaxa clar definită fac codul să fie mai citibil.
- *Ușor de menținut*
- *O bibliotecă standard largă* - Cea mai mare parte a bibliotecii Python este foarte portabilă și compatibilă cu mai multe platforme (linux, windows, etc.)
- *Mod interactiv* - Python are suport pentru un mod interactiv care permite testarea interactivă și depanarea fragmentelor de cod.
- *Portabil* - Python poate rula pe o mare varietate de platforme hardware și are aceeași interfață pe toate platformele.
- *Extensibil* – Se pot adăuga adăuga module scrise într-un limbaj de nivel mai jos la interpretorul de Python. Aceste module permit programatorilor să adauge sau să își personalizeze instrumentele pentru a fi mai eficienți.
- *Scalabil* - Python oferă o structură mai bună și suport pentru programe mari decât pentru scripturi.

2.5 Biblioteci folosite

În cadrul proiectului, am folosit următoarele biblioteci:

- *google.colab*, prin intermediul căreia am putut monta fișierele din Google Drive în enviromentul de Google Colab. Datorită dimensiunii mari a setului de date folosit, cât și a pașilor intermediari și a rezultatelor salvate, Google Drive a reprezentat soluția optimă de a depozita datele necesare, în special datorită funcționalității oferite de biblioteca menționată.
- *xml.etree.ElementTree* reprezintă biblioteca folosită pentru a parsa și a extrage datele din documentele de tip XML.
- *numpy* reprezintă o bibliotecă care oferă posibilitatea de a lucra cu date numerice de tip vectorial. Deși python oferă deja un tip de date care poate fi folosit în acest scop, adică listele, numpy oferă numeroase avantaje, cum ar fi faptul că datele sunt stocate într-o zonă continuă de memorie, iar asta face prelucrarea unui șir de numere salvate într-un numpy array mult mai ușoară.
- *nltk (Natural Language Toolkit) o*feră interfețe ușor de utilizat la peste 50 de corpuri și resurse lexicale, împreună cu o suită de biblioteci de procesare a textului pentru clasificare, tokenizare, stemming, etichetare, analiză și raționament semantic. În cadrul proiectului, m-am folosit de corpusul folosit de această bibliotecă pentru cuvintele de tip stop word.
- *math* oferă acces la funcțiile matematice definit de standardul C.
- *regex* oferă operații necesare pentru folosirea și prelucrarea expresiilor regulate.
- *pickle* implementează protocoale binare pentru serializarea și de-serializarea unei structuri de obiecte Python. Această bibliotecă a fost de ajutor de-a lungul dezvoltării proiectului pentru salvarea în memoriei a obiectelor ce reprezentau pași intermediari în rularea pipeline-ului.
- *sentece_transformers* oferă accesul la un framework pentru crearea de embeddings pentru text. Prin intermediul acestei biblioteci am folosit modelul pre-antrenat de tip transformer într-un pas din proiect.
- m*atplotlib* este o bibliotecă folosită pentru a crea reprezentări și manipula reprezentări grafice.
- *sklearn* oferă o serie largă de funcționalități necesare pentru a crea programe folosind metode de învățare automată, cum ar fi algoritmi de clasificare, regresie, clustering și așa mai departe.

- *subprocess* oferă funcționalitatea de a executa procese de sistem și prelua datele rezultat oferite de procesul rezultat.

- *glob* este utilizat pentru a găsi toate căile de acces care corespund unui model specificat în conformitate cu regulile utilizate de shell-ul Unix. Rezultatele sunt returnate în ordine arbitrară.

- *copy* oferă operații generice pentru shallow copy și deep copy.

Pe lângă bibliotecile menționate, m-am folosit de codul pentru Encoplot pus la dispoziție în lucrarea *ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection,* realizată de Cristian Grozea, Christian Gehl și Marius Popescu, în care am modificat doar forma de afișare a rezultatului. Am compilat codul folosindu-mă de *GNU c++ compiler,* iar folosirea obiectului rezultat am făcut-o prin intermediul modulului *subprocess*, menționat în enumerația anterioară.

2.6 Google Colab

Colaboratory, sau „Colab” pe scurt, este un produs Google Research. Colab permite oricui să scrie și să execute coduri Python, în mode arbitrar, prin browser și este deosebit de potrivit pentru domeniul învățării automate, analiza datelor și domeniul academic. Într-un mod mai tehnic, Colab este un serviciu de notebook-uri Jupyter găzduit care nu necesită nicio configurare pentru utilizare, oferind în același timp acces gratuit la resurse de calcul, inclusiv GPU și TPU.

Codul dezvoltat în cadrul Google Colab este executat într-o mașină virtuală privată asociată cu contul folosit. Mașinile virtuale sunt șterse când sunt în repaus pentru o perioadă de timp și au o durată de viață maximă impusă de serviciul Colab.

Colab este capabil să ofere resurse gratuite, parțial, având limite de utilizare dinamice, care uneori fluctuează. Astfel, ținând cont de limitele generale de utilizare, precum și perioadele de expirare datorata inactivității, durata maximă de viață a unei mașini virtuale, tipurile de GPU disponibile și alți factori variază în timp. Colab nu publică aceste limite, în parte, deoarece acestea pot (și uneori o fac) să se schimbe în timp rapid.

GPU-urile și TPU-urile sunt uneori prioritare pentru utilizatorii care folosesc Colab interactiv decât pentru rulările de lungă durată, sau pentru utilizatorii care au folosit recent mai puține resurse în Colab. Drept urmare, utilizatorii care folosesc Colab pentru rulări de lungă durată sau utilizatorii care au folosit recent mai multe resurse în Colab, sunt mai predispuși să se confrunte cu limite de utilizare și să aibă acces temporar la GPU-uri și TPU-uri. Limite de utilizare mai mari și mai stabile pot fi oferite de Colab Pro.

Printre avantajele Google Colab se numără următoarele:

- Posibilitatea de a salva și a importa notebook-uri din Google Drive sau Github
- Diverse biblioteci și frameworks integrate, precum PyTorch, TensorFlow, Keras, OpenCV, etc. De asemenea, instalarea bibliotecilor se face ușor, prin intermediul pip (package installer for Python).
- Oferă opțiunea de „colaborare”, mai mulți programatori putând să lucreze la același notebook în același timpul
- Codul este executat pe mașini virtuale, cu acces la GPU sau TPU

În imaginea următoare se pot observa adăugările lui Google Colab față de Jupyter Notebook și iPython.

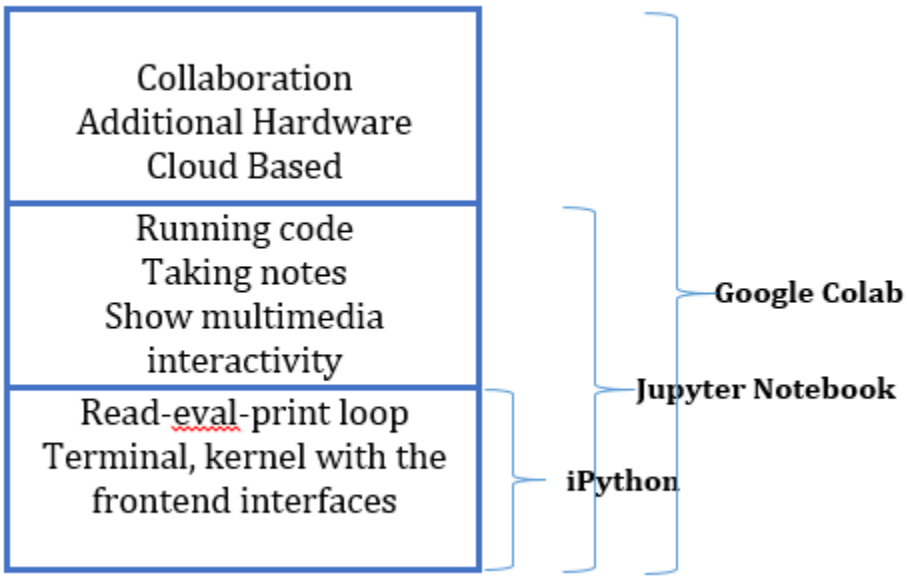


Figura 3: Evoluția iPython, Jupyther și Colab

2.7 ENCOPLOT

Encoplot (N-Gram Coincidence Plot) reprezintă soluția folosită de către Cristian Grozea, Christian Gehl și Marius Popescu în lucrarea *ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection*.

În domeniul procesării de limbaj natural, un N-gram este o secvență contiguă de N elemente dintr-un eșantion dat de text. Elementele pot fi foneme, silabe, caractere, cuvinte sau perechi de bază în funcție de aplicație. În cadril Encoplot s-au folosit N-grame de caractere.

Algoritmul are următorii pași:

Date de intrare: Două șiruri de caractere, A și B, pentru a fi comparate

Date de ieșire: O listă de perechi (x, y) de poziții din A, respectiv B, unde se află exact același N-gram.

1. Se extrag N-gramele din A și din B
2. Se sortează cele două liste de N-grame
3. Se compară cele două liste de prin intermediul unui algoritm de merge sort modificat. Când două N-grame, cu cea mai mică poziție în cele două șiruri, sunt egale, atunci se adaugă în datele de ieșire poziția în A și în B.

Ca și exemplu, se pot considera următoarele șiruri de caracter:

A = abcabd B = xabdy

Pentru N = 2, rezultatele pentru Encoplot sunt descrise în tabelul următor:

Encoplot pairs	Dotplot pairs
1 2 ab	1 2 ab
	4 2 ab
5 4 bd	5 4 bd

Tabelul 2: Exemplu Encoplot vs. Dotplot

În tabelul de mai sus se află atât rezultatele pentru Encoplot, cât și rezultatele în cazul în care s-ar fi făcut o comparație de tip dotplot între cele două siruri de caractere, adică s-ar fi selectat fiecare pereche de N-grame identice. Spre deosebire de dotplot, Encoplot selectează doar prima potrivire a unui n-gram.

Pentru N = 3, rezultatele ar fi identice, pentru că nu ar exista N-grame care s-ar repeta, după cum se poate vede și în tabelul de mai jos.

Encoplot pairs	Dotplot pairs
4 2 abd	4 2 abd

Tabelul 3: Exemplu Encoplot vs. Dotplot

Printre avantajele Encoplot se numără:

- Timp linear garantat (O(n)), în timp ce dotplot are complexitate pătratică ($O(n^2)$).
- Poate fi folosit în diverse domenii
- Poate fi optimizat în așa fel încât să aibă o implementare foarte rapidă (cu N până la 16 pe calculatoarele cu procesoare pe arhitectura de 64 de biți)

Faptul că Encoplot preia doar prima apariție identică între cele două texte nu constituie neapărat un dezavantaj. Dacă un N-gram este duplicat de-a lungul unui text, asta poate să însemne că are o cantitate informațională redusă, putând fi constituit mai degrabă de formulări generice decât de formulări cu conținut informațional. De exemplu, structura *"despite this, we are"* poate apărea de mai multe ori de-a lungul unui text, însă nu oferă conținut informațional. Astfel, chiar dacă s-ar returna mai multe potriviri cu aceasta de-a lungul celor două texte, acestea nu ar fi relevante pentru problema plagiatului. În schimb, N-gramele unice de-a lungul unui text au șanse mari să conțină conținut informațional, astfel existând o șansă mai mare ca la potrivirea acestora în cele două texte să existe un caz de plagiat.

2.8 TRANSFORMER

Modelele de tip *transformer* sunt cea mai recentă intrare în lumea modelelor de învățăre profundă (deep learning) pentru procesarea de limbaj natural.

Modelele transformer au devenit de actualitate pentru aproape toate sarcinile majore procesare de limbaj natural în ultimii câțiva ani. Ele modelează contextul textului, dar nu într-o manieră secvențială. Având un cuvânt în intrare, preferă să privească toate cuvintele din jurul său (tehnică cunoscută sub numele de *auto-atenție*) și să reprezinte fiecare cuvânt în raport cu contextul său. De exemplu, cuvântul „bancă” poate avea semnificații diferite în funcție de contextul în care apare. Dacă contextul vorbește despre finanțe, atunci „bancă” denotă probabil o instituție financiară. Pe de altă parte, dacă contextul menționează un parc, atunci sensul cuvântului se schimbă total. Modelele transformer pot modela un astfel de context și, prin urmare, au fost utilizate intens în sarcinile de procesare de limbaj natural datorită acestei capacități de reprezentare mai mari în comparație cu alte modele bazate pe rețele neuronale.

Recent, modelele transformer au fost utilizate pentru transferul de învățare pentru diferite sarcini de procesare de limbaj natural. Învățarea prin transfer este o tehnică în AI în care cunoștințele acumulate în timpul rezolvării unei probleme sunt aplicate unei probleme diferite, dar conexe. Cu modelele transformer, ideea este de a antrena un model foarte mare într-o manieră nesupravegheată (cunoscută sub numele de pre-antrenament) pentru a prezice o parte a unei propoziții având în vedere restul conținutului, astfel încât să poată codifica nuanțele la nivel înalt ale limbajului în ea. Aceste modele sunt antrenate pe mai mult de 40 GB de date textuale, extrase de pe întregul internet.

Un exemplu de model transformer este BERT (Bidirectional Encoder Representations from Transformers), prezentat în figura de mai jos, care este pre-antrenat pe date masive și open-source de către Google.

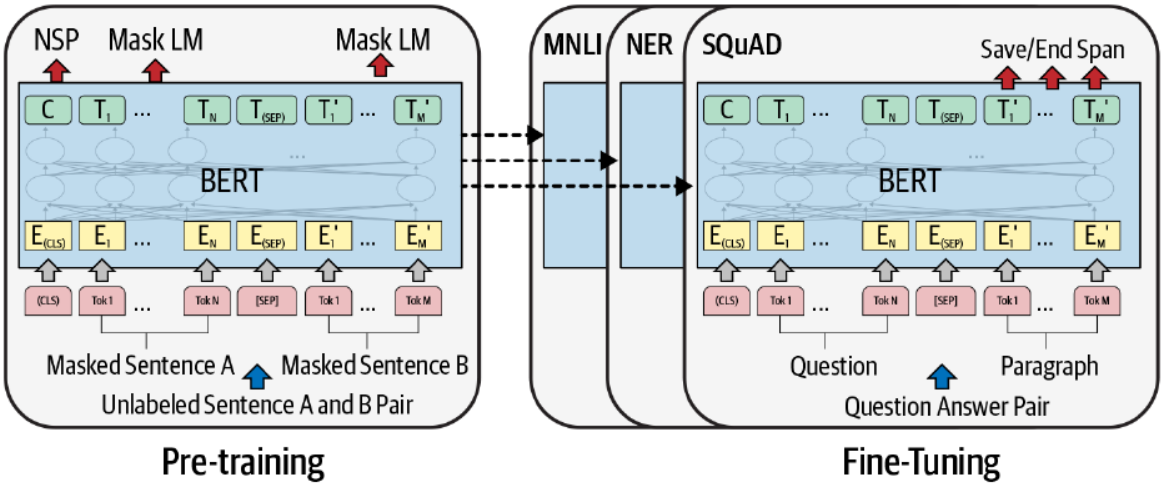


Figura 4: Exemplu model de transformer Bert

Modelul pre-antrenat este prezentat în partea stângă a figurii. Acest model este apoi modificat prin fine-tuning pentru diferite sarcini de procesare de limbaj natural, cum ar fi clasificarea textului, răspunsuri la întrebări etc. Datorită cantității de cunoștințe pre-antrenate, BERT lucrează eficient la transferul de cunoștințe pentru sarcinile necesare și și oferă rezultate excelente pentru multe dintre acestea.

În figura următoare se ilustrează funcționarea unui mecanism de auto-atenție, care este o componentă cheie a unui model transformer.

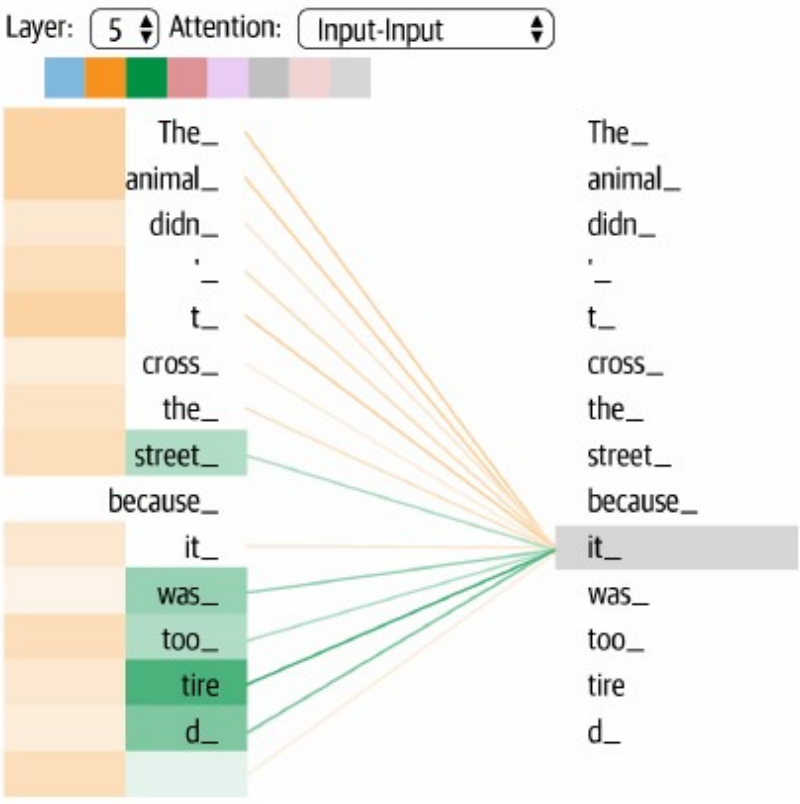


Figura 5: Mecanism de auto-atenție

2.9 SIMILITUDINEA COSINUS

Similitudinea cosinus este o metrică utilizată pentru a determina cât de asemănătoare sunt două documente (şiruri de caractere), indiferent de dimensiunea lor.

Matematic, măsoară cosinusul unghiului dintre cei doi vectori proiectați într-un spațiu multidimensional şi determină dacă cei doi vectori sunt orientați către aceeaşi direcție. În acest context, cei doi vectori sunt reprezentări ale documentelor de intrare, adică matrici care conțin numărul de cuvinte din două documente sau rezultatele de tip word embeddings obținute prin intermediul unui transformer.

Un document poate fi reprezentat de mii de atribute, fiecare înregistrând frecvența unui anumit cuvânt (cum ar fi un cuvânt cheie) sau o expresie din document. Astfel, fiecare document este un obiect reprezentat de ceea ce se numeşte vector de termen-frecvență. De exemplu, în tabelul de mai jos, vedem că Documentul 1 conține cinci instanțe ale cuvântului „team", în timp ce „hockey" are loc de trei ori. Cuvântul „coach" este absent din întregul document, după cum se indică printr-o valoare de numărare 0. Aceste date pot fi extrem de asimetrice.

Document	**team**	**coach**	**hockey**	**baseball**	**soccer**	**penalty**	**score**	**win**	**loss**	**season**
Doc1	5	0	3	0	2	0	0	2	0	0
Doc2	3	0	2	0	1	1	0	1	0	1
Doc3	0	7	0	2	1	0	0	3	0	0
Doc4	0	1	0	0	1	2	2	0	3	0

Tabelul 4: Exemplu de reprezentări vectoriale pentru documente text

Vectorii de termen-frecvență sunt de obicei foarte lungi şi rarefiați (adică au multe valori 0). Măsurile tradiționale de distanță nu funcționează bine pentru date numerice atât de rarefiate. De exemplu, doi vectori de termen-frecvență pot avea multe valori de 0 comune, ceea ce înseamnă că documentele corespunzătoare nu împărtăşesc multe cuvinte, dar acest lucru nu le face similare. Avem nevoie de o măsură care să se concentreze pe cuvintele pe care cele două documente le au în comun şi pe frecvența apariției a acestor cuvinte. Cu alte cuvinte, avem nevoie de o măsură pentru datele numerice care ignoră potrivirile de 0.

Similitudinea cosinus este o măsură a similitudinii care poate fi utilizată pentru a compara documente sau, în mod echivalent, pentru a da o clasificare a documentelor în raport cu un vector dat de cuvinte de interogare. Fie x și y doi vectori pentru comparație. Definim următoarea funcție:

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

unde, $\|x\|$ este norma euclidiană a vectorului $x_1^2 + x_2^2 + \ldots + x_p^2$, definită ca $\sqrt{x_1^2 + x_2^2 + \ldots + x_p^2}$. Ca și concept, acesta reprezintă lungimea vectorului. În mod similar, $\|y\|$ este norma euclidiană a vectorului **y**. Similitudinea cosinus calculează unghiul dintre vectorii **x** și **y**. O similitudine cosinus cu valoarea 0 înseamnă că cei doi vectori se află la unghi de 90 de grade, adică nu există similitudine între cei doi vectori. Cu cât valoarea similitudinii cosinus este mai aproape de 1, cu atât unghiul dintre cei doi vectori este mai mic, deci similitudinea este mai mare.

Presupunând că folosim datele corespunzătoare primelor două documente din tabel pentru a calcula similitudinea cosinus pe baza formulei prezentate anterior. Astfel, primul document ar corespunde vectorului $x = (5,0,3,0,2,0,0,2,0,0)$, iar al doilea document ar corespunde vectorului $y = (3,0,2,0,1,1,0,1,0,1)$.

Produsul celor doi vectori va fi:

$$x^t \cdot y = 5 \times 3 + 0 \times 0 + 3 \times 2 + 0 \times 0 + 2 \times 1 + 0 \times 1 + 0 \times 0 + 2 \times 1 + 0 \times 0 + 0 \times 1 = 25$$

Norma euclidiană a vectorului x se va calcula astfel:

$$\|x\| = \sqrt{5^2 + 3^2 + 2^2 + 2^2} = \sqrt{42} \simeq 6.42$$

Norma euclidiană a vectorului y se va calcula astfel:

$$\|y\| = \sqrt{3^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2} = \sqrt{42} \simeq 4.12$$

Astfel, valoarea similitudinii cosinus va fi următoarea:

$$\text{sim}(x, y) \simeq \frac{25}{6.42 \cdot 4.12} \simeq 0.94$$

Ținând cont de rezultatul anterior, putem considera că documentele exemplu sunt similare.

2.10 DBSCAN

DBSCAN (Density-based spatial clustering of applications with noise) este un algoritm de clustering a datelor propus de Martin Ester, Hans-Peter Kriegel, Jörg Sander şi Xiaowei Xu în 1996. Este un algoritm non-parametric de clustering bazat pe densitate: având în vedere un set de puncte într-un anumit spațiu, grupează punctele care sunt dense (puncte cu mulți vecini din apropiere), marcând ca puncte aberante punctele care se află singure în regiunile cu densitate redusă (ai căror vecini cei mai apropiați sunt prea departe). DBSCAN este unul dintre cei mai comuni algoritmi de clustering și, de asemenea, printre cei mai citați în literatura științifică.

Luându-se în considerare un set de puncte dintr-un spațiu care trebuie grupat, fie ε un parametru care specifică raza unei vecinități față de un anumit punct. În scopul procesului de clustering DBSCAN, punctele sunt clasificate ca puncte de bază, puncte (dens-)accesibile și puncte aberante, după cum urmează:

- Un punct p este un punct de bază dacă cel puțin *minPts* puncte sunt la distanța ε de acesta (inclusiv p), unde *minPts* este numărul minim de puncte.
- Un punct q este direct accesibil de la p dacă punctul q se află la distanța ε de punctul de bază p. Se spune că punctele sunt direct accesibile din punctele de bază.
- Un punct q este accesibil din p dacă există o cale p_1, ..., p_n cu $p_1 = p$ și $p_n = q$, unde fiecare p_{i+1} este direct accesibil din p_i. Acest lucru implică faptul că punctul inițial și toate punctele de pe cale trebuie să fie puncte de bază, cu posibila excepție a lui q.
- Toate punctele care nu pot fi accesate din niciun alt punct sunt puncte aberante sau puncte de zgomot.

Dacă p este un punct de bază, atunci formează un cluster împreună cu toate punctele (de bază sau nu) care sunt accesibile din acesta. Fiecare cluster conține cel puțin un punct central; punctele care nu sunt de bază pot face parte dintr-un cluster, dar formează „marginea” acestuia, deoarece nu pot fi folosite pentru a atinge mai multe puncte.

Accesibilitatea nu este o relație simetrică: prin definiție, numai punctele de bază pot atinge puncte care nu sunt de bază. Opusul nu este adevărat, deci un punct care nu este de bază poate fi atins, dar nu se poate ajunge la nimic din acesta. Prin urmare, este necesară o noțiune suplimentară de conectivitate pentru a defini în mod formal întinderea clusterelor găsite de DBSCAN. Două puncte p și q sunt dens-conectate dacă există un punct o astfel încât atât p cât și q sunt accesibile de la o. Dens-conectivitatea este simetrică.

Un cluster îndeplinește apoi două proprietăți:

1. Toate punctele din cluster sunt dens-conectate reciproc.
2. Dacă un punct este dens-accesibil dintr-un punct al clusterului, acesta face parte și din cluster.

În figura de mai jos, *minPts* are valoarea 4. Punctul A și celelalte puncte roșii sunt puncte centrale, deoarece zona care înconjoară aceste puncte într-o rază ε conține cel puțin 4 puncte (inclusiv punctul în sine). Deoarece toate sunt accesibile unul de altul, formează un singur cluster. Punctele B și C nu sunt puncte de bază, dar sunt accesibile de la A (prin alte puncte de bază) și aparțin și clusterului. Punctul N este un punct de zgomot care nu este nici un punct central, nici accesibil direct.

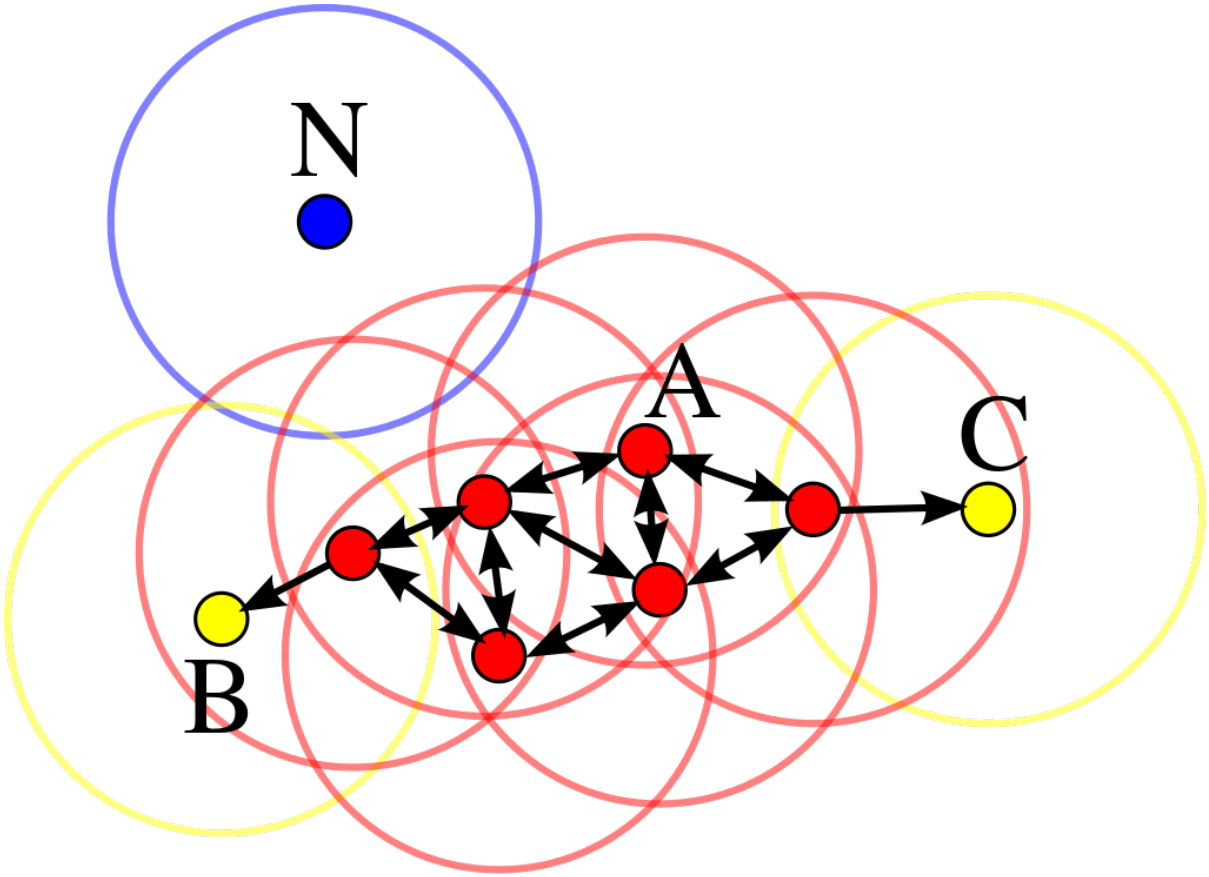


Figura 6: Exemplu DBSCAN

3 DESIGN ŞI IMPLEMENTARE

3.1 Descrierea setului de date

Pentru a putea verifica şi valida implementarea, m-am folosit de setul de date *PAN-PC-09.* Acesta pune la dispoziție *41 223* de documente text în care au fost inserate în mod artificial *94 202* cazuri de plagiat artificial.

Corpusul este extras din cărți din cadrul *Project Gutenberg.* Astfel, pentru a fi realizat s-au folosit *22 135* de cărți scrise în Engleză, *527* de cărți scrise în Germană şi 211 cărți scrise în Spaniolă.

Corpusul a fost realizat pentru a putea fi folosit pentru două tipuri de probleme în cadrul detecției de plagiat:

- Plagiat intrinsec – Dându-se un set de documente suspicioase, cerința este de a determina pasajele plagiate, fără a se face comparație între documente. (de exemplu, să se determine pasajele plagiate ținându-se cont de stilul folosit în cadrul documentului)
- Plagiat extern – Dându-se un set de documente suspicioase şi un set de documente sursă, cerința este de a determina pasajele plagiate din documentele suspicioase şi pasajele de text corespunzătoare din documentele sursa.

În cadrul acestei lucrări m-am axat pe abordarea celei de-a doua cerințe, respectiv detecția plagiatului extern. Astfel, din totalul de documente, am avut la dispoziție un set de *14 429* documente sursă şi *14 428* documente suspicioase. Din motive ce țin de performanță şi viitoare optimizări, am extras un subset al acestora, detaliu ce va fi explicat în cadrul capitolului de *Rezultate şi experimente*.

Corpusul este împărțit în două directoare: „*source-documents”* şi „*suspicious-documents”*.

Atât *source-documents*, cât şi *suspicious-documents* conțin 8 directoare de forma “*partX*” (unde X este numărul directorului).

Fiecare director sursă va conține următoarele tipuri de fişiere:

- „*source-documentKKKKK.txt”* – un fişier text care a fost folosit ca sursă pentru inserările de plagiat artificial din cadrul documentelor suspicioase. *KKKKK* reprezintă numărul documentului.

- „*source-documentKKKKK.xml*” – un fișier XML care oferă detalii despre fișierul text cu același nume: sursa documentului din cadrul Project Gutenberg și limba în care este scris.

Exemplu de fișier XML pentru document sursă:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.uni-weimar.de/medien/webis/research/corpora/pan-pc-09/document.xsd"
reference="source-document00001.txt">
  <feature name="project-gutenberg" etext_number="12517" url="http://www.gutenberg.org/files/12517/12517-8.txt"/>
  <feature name="language" value="en" />
</document>
```

Figura 7: Exemplu fișier XML pentru un document sursă

Fiecare director suspicios va conține următoarele tipuri de fișiere:

- „*suspicious-documentKKKKK.txt*”: un fișier text în care este posibil să se afle secțiuni plagiate artificial. La fel ca la documentele sursă, *KKKKK* reprezintă numărul documentului.
- „*suspicious-documentKKKKK.xml*” - un fișier xml care oferă detalii despre fișierul text cu același nume, în care pe lângă informațiile din cadrul fișierelor sursa, s-au adăugat următoarele caracteristici pentru fiecare secțiune plagiată:
    - *translation* (**true** sau **false**) – în cazul în care este vorba de plagiat lingvistic.
    - *obfuscation* (**none**, **low** sau **high**) - nivelul de „procesare” care a fost făcut asupra pasajului plagiat artificial.
    - *this_offset* (Ex: 44294) – Poziția caracterului de la care începe pasajul plagiat
    - *this_length* (Ex: 19843) - Dimensiunea pasajului plagiat
    - source_reference (Ex: source-document05152.txt) - Documentul sursă din care a fost extras pasajul original
    - *source_offset* (Ex: 14370) – Poziția caracterului de la care începe secțiunea care a fost folosită pentru plagiat în documentul sursă
    - *source_length* (Ex: 19880) - Dimensiunea pasajului din documentul sursă folosit pentru plagiat

Exemplu de fișier XML pentru document suspicios:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.uni-weimar.de/medien/webis/research/corpora/pan-pc-09/document.xsd"
reference="suspicious-document00001.txt">
  <feature name="project-gutenberg" etext_number="490" url="http://www.gutenberg.org/files/490/490.txt"/>
  <feature name="language" value="en" />
  <feature name="artificial-plagiarism" translation="false" obfuscation="low" this_offset="731" this_length="279"
  source_reference="source-document01409.txt" source_offset="20620" source_length="263" />
  <feature name="artificial-plagiarism" translation="false" obfuscation="low" this_offset="8767" this_length="769"
  source_reference="source-document01409.txt" source_offset="14003" source_length="785" />
</document>
```

Figura 8: Exemplu fișier XML pentru document suspicios

Au fost inserate două tipuri de plagiat:

- Plagiat lingvistic - documentul sursă este în germană sau spaniolă, iar pasajul plagiat a fost tradus. 10% din plagiatul inserat este de acest fel.
- Plagiat aleator – în care s-au aplicat aleator, cu un anumit nivel de modificare (**none**, **low** sau **high**), următoarele metode:
    - *Random Text Operations*: Dându-se pasajul din textul sursă, s-au amestecat, șters, adăugat sau înlocuit cuvinte sau fraze scurte. Cuvintele nou adăugate sau înlocuite din pasajul plagiat se fac cu alte secțiuni din documentul plagiat.
    - *Semantic Word Variation*: Dându-se pasajul din textul sursă, fiecare cuvânt e înlocuit cu sinonime, antonime, hiponime sau hipernime, alese aleator. Cuvântul e păstrat doar dacă nu există cuvinte cu care să fie înlocuit.
    - *POS - preserving Word Shuffling*: Dându-se o secțiune din textul sursă, se determină părțile de vorbire (***p**arts **o**f **s**peech - POS*). După aceea, pasajul plagiat este creat amestecând cuvintele între ele, dar păstrând aceeași secvență de părți de vorbire.

Raportul legat de dimensiunile documentelor este de 50% documente de dimensiune mică (1 – 10 pagini), 35% documente de dimensiune medie (10 – 100 pagini), 15% documente de dimensiune mare (100 – 1000 pagini).

Jumătate din documentele suspicioase nu sunt plagiate, acestea neavând niciun pasaj în care să fi fost adăugat plagiat artificial. Cealaltă jumate are un nivel de plagiat între 0% și 100%. Histograma de mai jos prezintă distribuția raportată la nivelul de conținut plagiat:

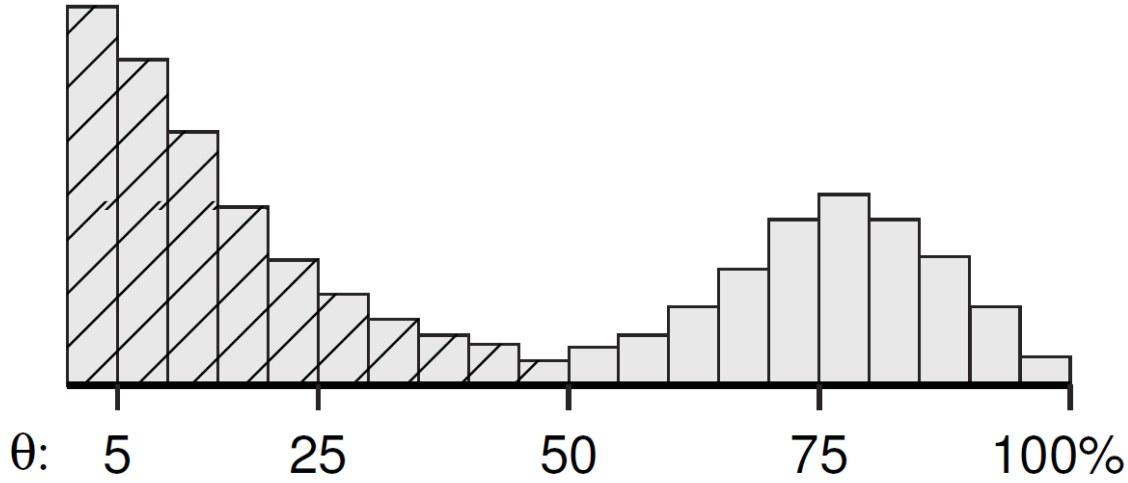


Figura 9: Histograma cu procentul de conținut plagiat pentru documentele suspicioase

Un pasaj plagiat poate fi de dimensiuni multiple, de la câteva propoziții, până la câteva pagini. Astfel, un pasaj plagiat poate avea între 50 și 5000 de cuvinte.

3.2 Descrierea metricilor de validare

Pentru a putea explica modul în care am implementat metricile de evaluare, o să mă raportez la figura de mai jos.

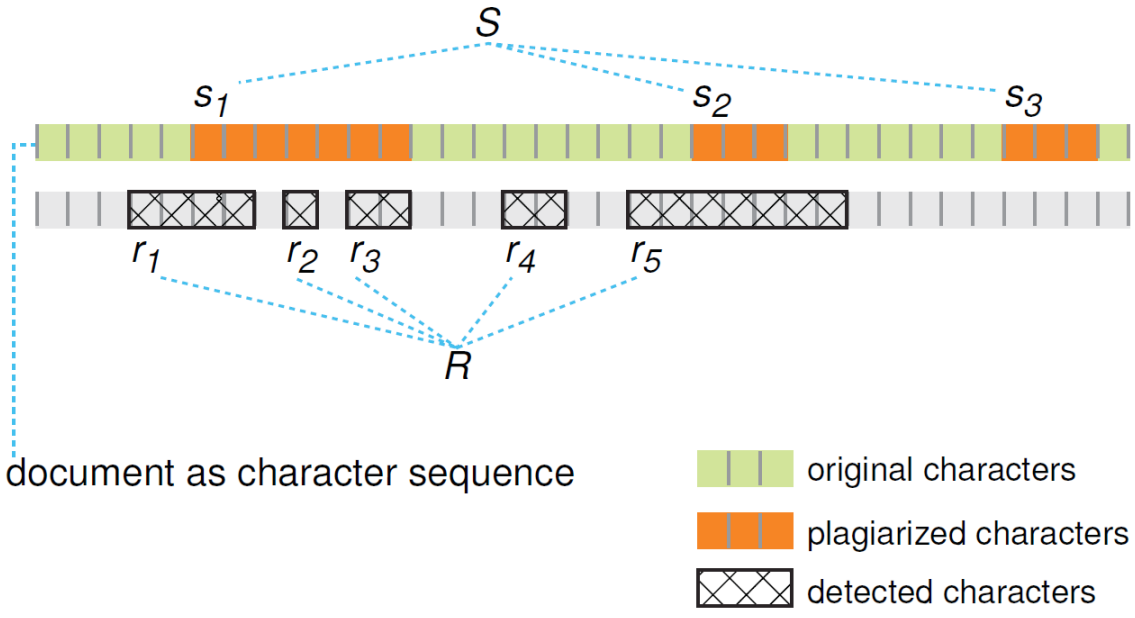


Figura 10: Exemplu de document plagiat și predicție

În imagine, documentul suspicios conține 3 pasaje plagiate: s_1, s_2 și s_3, toate formând mulțimea *S*.

Pasajele detectate ca fiind plagiate plagiate sunt reprezentate de mulțimea *R*, care conține pe r_1, r_2, r_3, r_4 și r_5.

Dacă s-ar calcula valoarea de reapel, de tip micro-averaged, atunci se poate observa că ar rezulta valoarea *8/13*. (8 caractere true positive / 13 caractere plagiate)

De asemenea, valoarea preciziei, de tip micro-averaged, ar fi *8/16*. (8 caractere true positive / 16 caractere detectate)

Avantajul unei astfel de metode este faptul ca este foarte ușor computabilă. Dezavantajul ar fi, însă, faptul că nu ține cont de mărimea secțiunii. În acest caz, mărimea secțiunii este corelata cu dificultatea de a fi detectată, dat fiind că fiecare caracter omis din secțiune ar fi interpretat ca o penalizare. O secțiune mai mare ar însemna mai multe caractere care trebuie detectate.

Soluția este de a calcula valoarea reapel și preciziei ca fiind orientate pe sistem, adică ca valori macro-averaged.

Pentru a putea calcula reapelul macro-averaged, se definește următoarea funcție:

$$R_{macroaveraged}(S,R) = \frac{1}{|S|}\sum_{s \in S}\frac{|s \nabla U_{r \in R'}|}{|s|}$$

unde ∇ calculează numărul de caractere care se suprapun pozițional cu **s** și oricare predicție **r** din **R**. Astfel, pentru fiecare pasaj **s** se calculează reapelul realizat pe baza lui (numărul de caractere din pasaj care se suprapun cu detecția / numărul de caractere total al lui **s**), după care se împarte la numărul total de pasaje plagiate. Rezultatul este, astfel, media reapelului fiecărui pasaj, fiecare pasaj având o pondere egală în reapelul final.

În cazul preciziei, se definește următoarea funcție:

$$P_{macroaveraged}(S,R) = \frac{1}{|R|}\sum_{r \in R}\frac{|r \nabla U_{s \in S'}|}{|r|}$$

Astfel, pentru fiecare pasaj **r** se calculează precizia realizată pe baza lui (numărul de caractere din pasaj care se suprapun cu pasajul sursă / numărul de caractere total al lui **r**), după care se împarte la numărul total de detecții. Rezultatul este, astfel, media preciziei fiecărui pasaj, fiecare pasaj având o pondere egală în precizia finală.

O problemă comună atât pentru modalitățile micro-averaged și macro-averaged este faptul că nu se ține cont de numărul de segmente diferite detectate pentru o singură secțiune plagiată. Astfel, se definește granularitatea, care este descrisă de următoarea funcție:

$$gran(S,R) = \frac{1}{|S_S|} \sum_{s \in S_S} |C_s|$$

Unde $\mathbf{C_s}$ reprezintă numărul de detecții care se suprapun pentru pasajul **s**, iar $\mathbf{S_R}$ reprezintă mulțimea pasajelor s pentru care există cel puțin o detecție cu care se suprapune. Granularitatea reprezintă media valorilor $\mathbf{C_s}$ pentru fiecare pasaj **s** din $\mathbf{S_R}$. Astfel, dacă pentru un segment plagiat s-ar detecta trei segmente, granularitatea pentru acel segment va fi egală cu trei. Granularitatea totală pe de altă parte, înseamnă mediat totală a numărului de detecții per pasaj plagiat (cu condiția de a fi minim o detecție).

Scorul total va fi descris de formula de mai jos:

$$overall\_score(S,R) = \frac{F}{\log_2(1 + gran)}$$

Scorul total este egal cu raportul dintre F (care reprezintă scorul F1, calculat pe baza valorilor macro-averaged a preciziei și reapelului) și logaritm în baza 2 din 1 + granularitatea calculată pe baza formulei descrise înainte.

3.3 Realizarea pipeline-ului

Pipeline-ul este descris de următoarea figură:

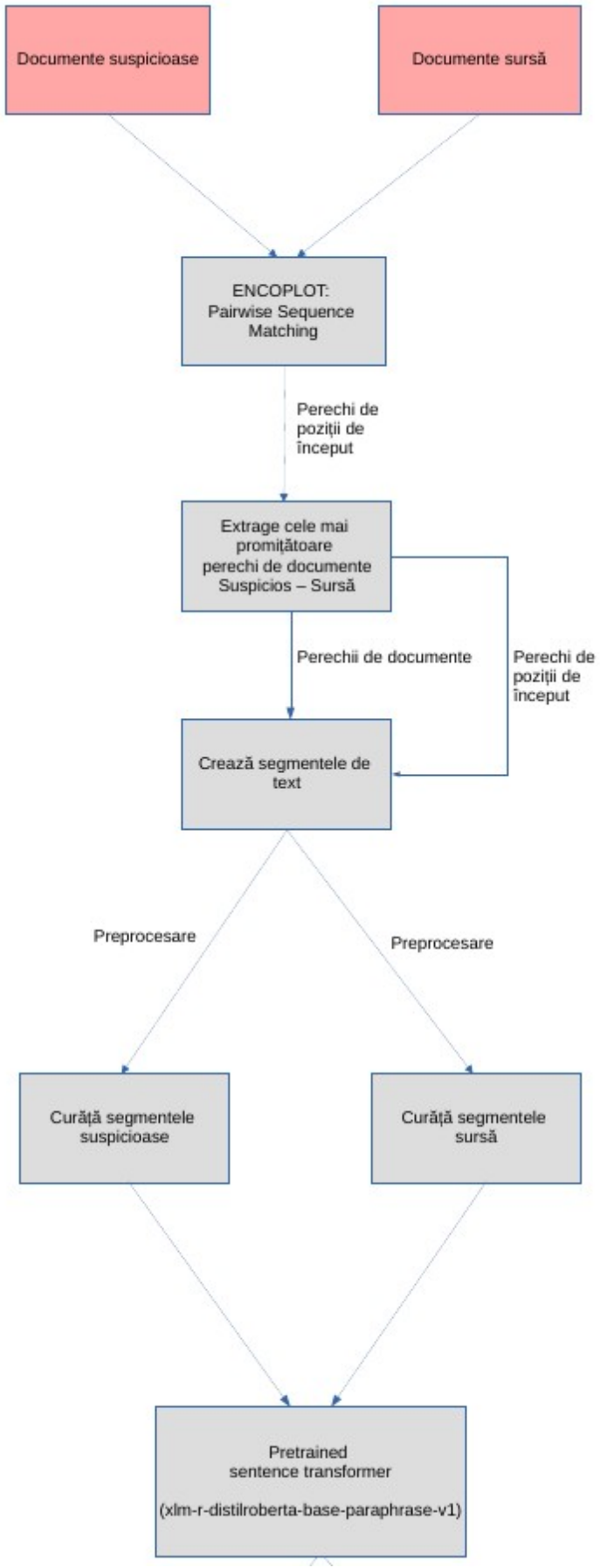


Figura 11: Prima parte a pipeline-ului

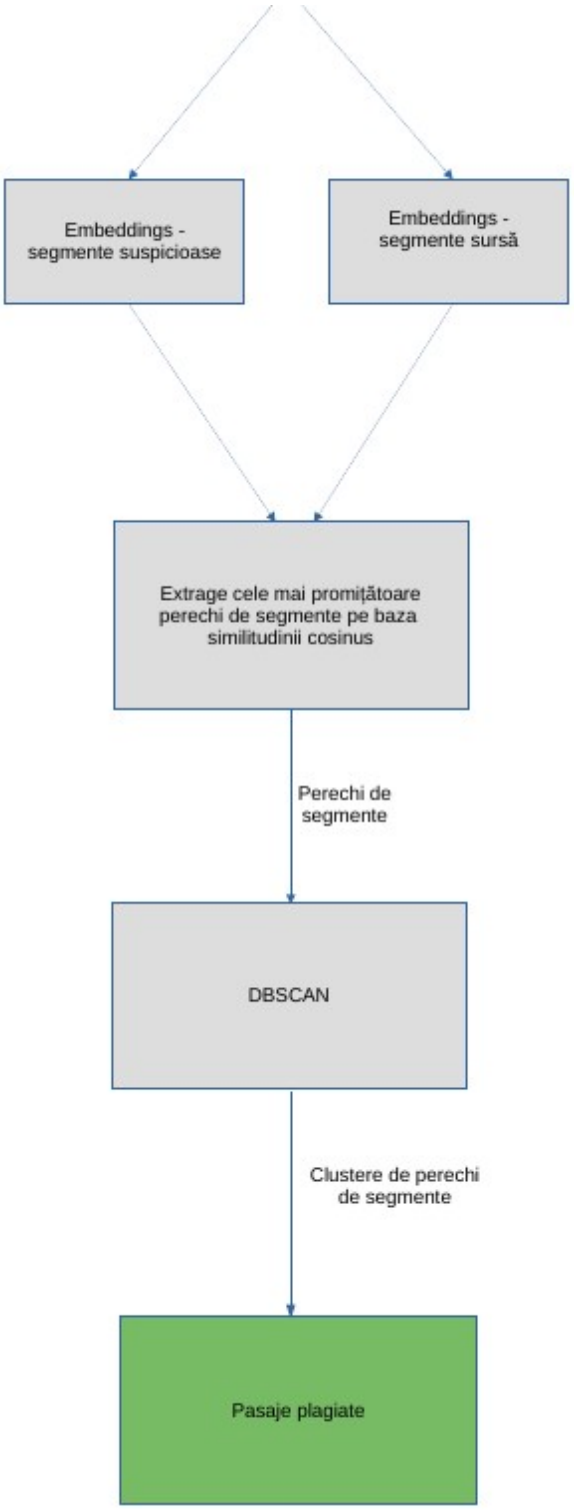


Figura 12: A doua parte a pipeline-ului

Așa cum se observă în figură, valorile de intrare sunt reprezentate de documentele suspicioase și de documentele sursă. Pentru a parcurge pipeline-ul, am ales să selectez câte un segment suspicios, pe care îl folosesc ca intrare pe rând alături de fiecare document sursă din setul de date.

Rezultatul este reprezentat de segmentele detectate ca fiind plagiate (caracterizate de poziție și document sursă).

Implementarea a fost realizată în Google Colab, iar codul sursă a fost structurat în celule într-un fișier de tip notebook. Am împărțit celulele în următoarele capitole:

- ***Imports and parameters*** – În acest capitol, codul din celule este responsabil de importarea bibliotecilor necesare, setarea parametrilor și compilarea codului necesare pentru Encoplot.
- ***Segment class for generic segments*** – În acest capitol se află celula în care este definită clasa Segment, care corespunde unui segment de text dintr-un document (sursă sau suspicios). De asemenea, o instanță acestei clase va conține textul neprocesat, textul pre-procesat, poziția și dimensiunea segmentului.
- ***Plagiarism Class - for document known plagiarism list - used for later evaluation*** – În acest capitol se află celula în care este definită clasa Plagiarism, care are rolul de a salva informațiile din caracteristica unui pasaj plagiat din documentele XML caracteristic fiecărui document. O instanță acestei clase va conține poziția în documentul suspicios, dimensiunea pajului în documentul suspicios, documentul sursă din care a fost extras, poziția în documentul sursă și dimensiunea pasajului în documentul sursă.
- ***FeatureList class - used for document known features*** – În acest capitol este definită clasa FeatureList, responsabila de definirea caracteristicilor documentului curent. Aceasta conține o listă de plagiarism (reprezentat de instanțe ale clasei definite în punctul anterior) și limba documentului pe care îl reprezintă. Pentru un document sursă, lista de plagiarism va fi goală.
- ***Document class - containing document features, segments, and embeddings*** - În acest capitol este definită clasa Document, care descrie un document (suspicios sau sursă). O instanță a acestei clase va conține textul documentului curent, lista de caracteristici al documentului curent (de tip FeatureList), numele documentului, segmentele candidat și embedding-urile pentru segmentele candidat.
- ***PredictedSegment class - used to store the most similar segment pairs by cosine similarity*** – În acest capitol se află definită clasa PredictedSegment, care descrie o predicție a pipeline-ului. O instanță a acestei clase va conține poziția pasajului în documentul suspicios,

dimensiunea pasajului în documentul sursă, numele documentului sursă, poziția pasajului în documentul sursă și dimensiunea pasajului în documentul sursă.

- ***get_most_similars(...) and plot methods - used to get the most similar segments pairs (due to cosine similarity)*** – În acest capitol se află celulele în care este definită funcția get_most_similars, care pe baza segmentelor candidat din documentul sursă și documentul suspicios și a similitudinii cosinus dintre aceste segmente de text, returnează o nouă listă de segmente candidat. De asemenea, în acest capitol se află și alte funcții responsabile de reprezentarea grafică a predicțiilor și a segmentelor extrase.

- ***Predictions class - used to compute the segments extractions, DBSCAN and the prediction result –*** În acest capitol se află celula în care este definită clasa *Predictions*, responsabil de calcularea predicțiilor finale.

- ***Encoplot integration - Used to get the Encoplot results and to compute a score based on the consecutive pairs –*** În acest capitol se află celula în care este definită clasa Encoplot responsabilă de apelarea obiectului obținut din compilarea codului pentru Encoplot, extragerea rezultatelor și calcularea unui scor bazat pe perechile de poziții consecutive.

- ***scan_for_plagiarism(...) and other usefull functions to parse the corpus*** – în celulele din acest capitol se află funcția *scan_for_plagiarism*, care poate fi folosită pentru a aplica pipeline-ul pentru un document suspicios și o listă dată de documente sursă și returnează un obiect de tip *Predictions* cu rezultatele curente. De asemenea, în acest capitol se află și alte funcții utile pentru a parcurge corpusul.

- ***Validation metrics - Used to check the score for the predictions results*** – în celula din acest capitol se află clasa *ValidationMetrics*, responsabilă de evaluarea și afișarea predicțiilor obținute din pipeline.

- ***Running examples*** – În celulele din acest capitol se află două exemple pentru rularea pipeline-ului folosind funcțiile și clasele descrise anterior.

- ***Experiments*** – În celulele din acest capitol se află codul prin care s-au obținut rezultatele, exemplele și experimentele din capitolele următoare.

Pentru configurarea parametrilor pentru pipeline, m-am folosit de o celulă de cod din primul capitol, *Imports and parameters*.

```
SEGMENTS_SIZE = 100 # size in characters (without cleaning)
COUNT_NUMBER = 5 # the number of segments taking care of
TH_COSIN_VALUE = 0.75
SECTION_POSITION_MAX_DIF = 100
SEGMENTS_WINDOW_SIZE = SEGMENTS_SIZE / 4

ENCOPLOT_MAX_STEP_DIF = 10
ENCOPLOT_MIN_FOUND_CONSECUTIVE = 30

MODEL_NAME = 'xlm-r-distilroberta-base-paraphrase-v1'

DBSCAN_EPS = SECTION_POSITION_MAX_DIF * SECTION_POSITION_MAX_DIF / 1.5
DBSCAN_MIN_SAMPLES = 3

PATH_SOURCE = "/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-corpus/
source-documents/part"
PATH_SUSPICIOUS = "/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-
corpus/suspicious-documents/part"
SUSPICIOUS_DOCUMENT_PREFIX = "suspicious-document"
SOURCE_DOCUMENT_PREFIX = "source-document"

SHOW_DETAILS = False
```

Pașii pe care i-am folosit pentru determinarea segmentelor plagiate, modul în care i-am implementat și alegerea parametrilor vor fi puncte descrise în capitolele următoare.

3.3.1 Integrarea metodei ENCOPLOT și extragerea documentelor candidat

Prima operație pe care o voi face asupra documentelor primite ca input este cea de *Encoplot*

Codul necesar pentru acest pas a fost luat din lucrarea „*ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection*”. Pentru a putea folosi codul în Google Colab, l-am compilat folosind comanda de shell specifică („*g++ encoplot.cpp*”), după care am chemat executabilul rezultat folosind un apel de sistem și dându-i ca argumente locațiile documentelor.

Operațiile pe care le face Encoplot constau în împărțirea documentelor de intrare în token-uri unicate de tip **16-gram** de caractere (se păstrează doar prima apariție al unui 16-gram), și compararea acestora între cele două documente. Rezultatul acestui pas este reprezentat de perechile de poziții din documentele de intrare în care **16-gramele** au coincis.

Aceste perechi de poziții reprezintă datele de intrare pentru următorul pas, unde, pe baza acestora „filtrez” documentele sursă. Am considerat că o pereche de documente este promițătoare (are mai multe șanse de a fi plagiată) dacă exista cel puțin un șir de **30** de perechi consecutive care să aibă aproximativ același salt (cu o diferență de caractere mai mică de **10**) în cadrul documentului sursă și al documentului suspicios. Astfel, dacă o pereche de documente îndeplinește această condiție, pipeline-ul se va continua. În caz contrar, se va considera că nu există pasaje plagiate între documentele respective.

Rularea obiectului obținut din compilarea codului din Encoplot și realizarea selecției explicată în pasul anterior a fost implementată în clasa Encoplot.

```
class Encoplot:

 def __init__(self, suspicious_doc, source_doc):
  self.current_score = 0
  self.suspicious_posible_offsets = []
  self.source_posible_offsets = []

  # get the encoplot results
  encoplot_raw_result = str(subprocess.check_output(["./a.out", suspicious_doc, source_doc]))

  # process the results -> a list sorted by the suspicious offset
  encoplot_raw_result = encoplot_raw_result.replace(",'", "")
  encoplot_raw_result = encoplot_raw_result.replace("'", "")
  encoplot_raw_result = encoplot_raw_result.replace("b", "")

  if(len(encoplot_raw_result) == 0):
   return

  encoplot_list = encoplot_raw_result.split(",")
```

```
for it in range(0, len(encoplot_list)):
  encoplot_list[it] = encoplot_list[it].split("-")
  encoplot_list[it][0] = int(encoplot_list[it][0])
  encoplot_list[it][1] = int(encoplot_list[it][1])

# sort the pairs
sorting_key = lambda a : a[0]
encoplot_list.sort(key=sorting_key)

Succesfullsteps = 0
lastStep = [0, 0]

# check the "score" based on the consecutive pairs
for it in encoplot_list:
  if abs(it[0] - lastStep[0] - it[1] + lastStep[1]) < ENCOPLOT_MAX_STEP_DIF:
    Succesfullsteps += 1
  else:
    Succesfullsteps = 0

  if Succesfullsteps == ENCOPLOT_MIN_FOUND_CONSECUTIVE:
    self.current_score += 1

  lastStep = it

# it the score is better than 0, save the segments as well
if self.current_score > 0:
  for it in encoplot_list:
    self.suspicious_posible_offsets.append(it[0])
    self.source_posible_offsets.append(it[1])
```

3.3.2 Preprocesarea segmentelor

Segmentele de text din cadrul documentelor vor fi preluate din cadrul pozițiilor primite din Encoplot. De asemenea, acest pas se va executa doar pentru segmentele care au trecut de filtrarea efectuată prin analiza rezultatelor de la Encoplot.

Segmentele corespund cu poziția de început a rezultatului din Encoplot cu condiția de a exista o distanță minimă între pozițiile de început a două segmente consecutive. Astfel, se vor ignora rezultatele care nu respectă această condiție. Fiecare segment se construiește având dimensiunea **100** de caractere din textul original, iar distanța minimă dintre pozițiile de început ale segmentelor este de **25** de caractere.

După construirea segmentelor, acestea urmează să fie reprocesate. Preprocesarea segmentelor constă în:

- eliminarea oricărui caracter care nu se află în alfabet. (orice caracter diferit de **a – z** și **A - Z**) și modificarea tuturor literelor (unde este cazul) în litere mici.
- eliminarea cuvintelor de tip *stop word* specifice limbii în care este scris textul respectiv. Precum am detaliat în capitolul **3.1**, limba este unul dintre caracteristicile specifice fiecărui document, descris prin fișierul XML cu același nume. Pentru a determina cuvintele de tip *stop word* în funcție de limbă, m-am folosit de biblioteca *nltk*.

```
segment text:  by the following sketches when
originally printed in the Preston Chronicle, combined with a desire,
largely
cleaned text:   following sketches originally printed preston chronicle combined desire largely

segment text:  failed in the attempt, he can't help
it; if he has succeeded, he is thankful. No writer can suit
everybody;
cleaned text:   failed attempt help succeeded thankful writer suit everybody

segment text:  and sometimes genteely
maltreated; our parochial divinities, who preside over the fate of
the poor; our
cleaned text:   sometimes genteely maltreated parochial divinities preside fate poor
```

Figura 13: Exemplu de segmente curățate

Lista de segmente rezultată în urma preprocesării, specifică fiecărui document, constă într-o listă de obiecte care conțin următoarele:

- textul neprocesat corespunzător segmentului

- textul curățat
- poziția de început din documentul din care a fost extras
- dimensiunea segmentului

Am implementat preprocesarea segmentelor în cadrul funcției *split_segments* din cadrul clasei *Document*. Următoarea porțiune de cod arată operațiile realizate pentru preprocesarea segmentelor din documentul curent.

```
# clean the raw text segments
    stop_words = stopwords.words(self.features.language)
    cleaned_segments = []

    for seg in segment_list:
      seg_splits = (re.sub('[^a-zA-Z]', ' ', seg[0])).lower()
      seg_splits = seg_splits.split()
      cleaned_seg = ""
      for current_seg in seg_splits:
        if current_seg not in stop_words:
          cleaned_seg += " " + current_seg

      cleaned_segments.append(cleaned_seg)

    for raw_s, cleaned_s in zip(segment_list, cleaned_segments):
      self.segments.append(Segment(raw_s[0], raw_s[1], cleaned_s))
```

3.3.3 Calcularea valorilor de tip Embeddings şi determinarea similitudinii cosinus

Pentru a calcula valorile de tip *embeddings* modelul pre-antrenat, de tip *sentence-transformer, paraphrase-xlm-r-multilingual-v1*, oferit de către *Huggingface*. Acesta este o versiunea *multilingual* al lui *paraphrase-distilroberta-base-v1*, care a fost antrenat pe mai mult de *50* de limbi.

Am ales un model multilingual pentru a putea acoperi și plagiatul lingvistic. De asemenea, am ales modelul menționat în urma testării rezultatelor pe o serie de mai multe modele.

```
[[ 0.07260364  0.00056877  0.31039345 ...  0.10130894  0.02436822
   0.0669862 ]
 [ 0.03047445 -0.06894646  0.2519448  ...  0.20736735  0.03404507
   0.51988894]
 [-0.05821065  0.42196158  0.24628842 ...  0.3567069   0.04948767
  -0.01149768]
 ...
 [ 0.17497322 -0.07513033 -0.03785814 ...  0.04673236 -0.04523945
  -0.07490133]
 [ 0.04206114 -0.12432586 -0.02076861 ...  0.45941174 -0.11785059
   0.06155813]
 [-0.09409108  0.48965508  0.24686721 ...  0.08526424  0.19087349
  -0.14997381]]
```

Figura 14: Exemplu de embeddings pentru segmentele filtrate dintr-un document care a trecut prin procesul de la pasul anterior

Calcularea valorilor embeddings se realizează în metoda *compute_embeddings* din clasa *Document*.

```
def compute_embeddings(self, model_name):
  prep_text = []

  for it in self.segments:
    prep_text.append(it.cleaned_text)

  model = SentenceTransformer(model_name)
  self.embeddings = model.encode(prep_text)
```

Astfel, fiecărui segment îi corespunde o reprezentare vectorială, care pot reprezenta date de intrare pentru calcularea similitudinii cosinus. Valoare similitudinii cosinus poate reprezenta nivelul de similaritate a celor două reprezentări vectoriale a segmentelor de intrare.

Rezultatul similitudinii cosinus pentru două documente va fi reprezentat de o matrice de similitudine, care conține valorile similitudinii pentru fiecare segment sursă comparat cu fiecare segment suspicios. Matricea va avea un număr de coloane egal cu numărul de segmente suspicioase și un număr de rânduri egal cu numărul de segmente sursă. De exemplu, dacă s-ar compara doua documente, cu *10* segmente suspicioase și *12* segmente sursă, atunci matricea de similitudine va avea mărimea *12 x 10*.

```
suspicious_doc segments:  282
source_doc segments:  254
similarity matrix:  (254, 282)
[[ 0.3288074   0.12213624  0.19495979 ...  0.07995336  0.15076905
   0.04045319]
 [ 0.22863066  0.18644553  0.2567692  ...  0.2886155   0.12167113
   0.29442996]
 [ 0.18983841  0.06559961  0.12864777 ...  0.02140516  0.02957748
   0.13131686]
 ...
 [ 0.18551798  0.11344656  0.3503421  ...  0.24727592  0.00723946
   0.12207127]
 [ 0.11817143  0.10007912  0.10232185 ...  0.26289302  0.04369278
   0.07363424]
 [ 0.05369418 -0.01951784  0.03341074 ...  0.03571232 -0.01959988
   0.2516585 ]]
```

Figura 15: Exemplu de matrice de similaritate pentru două documente

Pentru a extrage segmentele candidat pentru următorul punct din pipeline, parcurg fiecare coloană (care reprezintă fiecare segment suspicios), sortez descrescător valorile similitudinii cosinus din coloana respectiva (valorile respective reprezentând similitudinea dintre segmentul suspicios curent și celelalte segmente sursă). Pe baza valorilor respective, pentru fiecare segment suspicios, formez perechi cu primele **5** segmente sursă. Lista de perechi formate reprezintă intrarea pentru următorul pas din pipeline.

```
source_ref: source-document04063.txt this_segment: offset: 531 length: 107 source_segment: offset: 7152 length: 105 similarity_value: 0.4408387
source_ref: source-document04063.txt this_segment: offset: 531 length: 107 source_segment: offset: 7059 length: 118 similarity_value: 0.3894726
source_ref: source-document04063.txt this_segment: offset: 531 length: 107 source_segment: offset: 37501 length: 102 similarity_value: 0.36466897
source_ref: source-document04063.txt this_segment: offset: 531 length: 107 source_segment: offset: 37501 length: 124 similarity_value: 0.36201417
source_ref: source-document04063.txt this_segment: offset: 531 length: 107 source_segment: offset: 14637 length: 113 similarity_value: 0.3461306
source_ref: source-document04063.txt this_segment: offset: 1862 length: 107 source_segment: offset: 34902 length: 108 similarity_value: 0.4566419
source_ref: source-document04063.txt this_segment: offset: 1862 length: 107 source_segment: offset: 37501 length: 124 similarity_value: 0.32452628
source_ref: source-document04063.txt this_segment: offset: 1862 length: 107 source_segment: offset: 33013 length: 116 similarity_value: 0.31786197
source_ref: source-document04063.txt this_segment: offset: 1862 length: 107 source_segment: offset: 15724 length: 106 similarity_value: 0.3118136
source_ref: source-document04063.txt this_segment: offset: 1862 length: 107 source_segment: offset: 15670 length: 104 similarity_value: 0.31036025
source_ref: source-document04063.txt this_segment: offset: 4867 length: 103 source_segment: offset: 8693 length: 110 similarity_value: 0.4484056
source_ref: source-document04063.txt this_segment: offset: 4867 length: 103 source_segment: offset: 8731 length: 116 similarity_value: 0.4310369
source_ref: source-document04063.txt this_segment: offset: 4867 length: 103 source_segment: offset: 32092 length: 103 similarity_value: 0.42487526
source_ref: source-document04063.txt this_segment: offset: 4867 length: 103 source_segment: offset: 4691 length: 102 similarity_value: 0.41615808
source_ref: source-document04063.txt this_segment: offset: 4867 length: 103 source_segment: offset: 15266 length: 115 similarity_value: 0.41361707
source_ref: source-document04063.txt this_segment: offset: 7412 length: 108 source_segment: offset: 7841 length: 103 similarity_value: 0.4716885
source_ref: source-document04063.txt this_segment: offset: 7412 length: 108 source_segment: offset: 9803 length: 104 similarity_value: 0.40198833
source_ref: source-document04063.txt this_segment: offset: 7412 length: 108 source_segment: offset: 11754 length: 104 similarity_value: 0.38678914
source_ref: source-document04063.txt this_segment: offset: 7412 length: 108 source_segment: offset: 31335 length: 105 similarity_value: 0.38024962
source_ref: source-document04063.txt this_segment: offset: 7412 length: 108 source_segment: offset: 7758 length: 104 similarity_value: 0.37452787
source_ref: source-document04063.txt this_segment: offset: 10397 length: 105 source_segment: offset: 8026 length: 107 similarity_value: 0.33678758
source_ref: source-document04063.txt this_segment: offset: 10397 length: 105 source_segment: offset: 7004 length: 122 similarity_value: 0.32796884
source_ref: source-document04063.txt this_segment: offset: 10397 length: 105 source_segment: offset: 31732 length: 108 similarity_value: 0.32480824
source_ref: source-document04063.txt this_segment: offset: 10397 length: 105 source_segment: offset: 7841 length: 103 similarity_value: 0.31514776
source_ref: source-document04063.txt this_segment: offset: 10397 length: 105 source_segment: offset: 7059 length: 118 similarity_value: 0.30404484
source_ref: source-document04063.txt this_segment: offset: 15244 length: 105 source_segment: offset: 31732 length: 108 similarity_value: 0.4532569
source_ref: source-document04063.txt this_segment: offset: 15244 length: 105 source_segment: offset: 8026 length: 107 similarity_value: 0.39156458
source_ref: source-document04063.txt this_segment: offset: 15244 length: 105 source_segment: offset: 32499 length: 108 similarity_value: 0.34892058
source_ref: source-document04063.txt this_segment: offset: 15244 length: 105 source_segment: offset: 12338 length: 107 similarity_value: 0.34175688
source_ref: source-document04063.txt this_segment: offset: 15244 length: 105 source_segment: offset: 32456 length: 105 similarity_value: 0.31684643
```

Figura 16: Exemplu de perechi de segmente formate între un document suspicios şi un document sursă

Construirea matricii de similitudine şi extragerea segmentelor candidat pe baza similitudinii cosinus este realizată în funcţia *get_most_similars*:

```
def get_most_similars(source_doc, suspicious_doc):
  segments = []
  similarity_matrix = cosine_similarity(source_doc.embeddings, suspicious_doc.embeddings)

  if SHOW_DETAILS == True:
    print("suspicious_doc segments: ", str(len(suspicious_doc.segments)))
    print("source_doc segments: ", str(len(source_doc.segments)))
    print("similarity matrix: ", similarity_matrix.shape)
    print(similarity_matrix)

  for it in range(len(similarity_matrix[0])):
    current_row = similarity_matrix[:, it]

    # sort the indexes by the cosine similarity of the embeddings
    similar_ix = np.argsort(current_row)[::-1]

    # save the first N similar segments by the cosine value
    for i in range(COUNT_NUMBER):
      try:
        segments.append(PredictedSegment(suspicious_doc.segments[it], source_doc.doc_name,
source_doc.segments[similar_ix[i]], current_row[similar_ix[i]]))
      except:
        # not enough segments
        pass

  return segments
```

3.3.4 Extragerea segmentelor candidat, DBSCAN şi determinarea pasajelor plagiate

În urma pasului anterior, rezultă un set de perechi de segmente sursă şi suspicioase şi valoarea similarității cosinus între embedding-urile acestor segmente. Dacă s-ar pune într-un plan în care axele ar fi formate din poziția de început a segmentului suspicios, poziția de început a segmentului sursă și valoarea similarității cosinus, s-ar putea observa „nişte” vârfuri, asemenea exemplului din figura de mai jos:

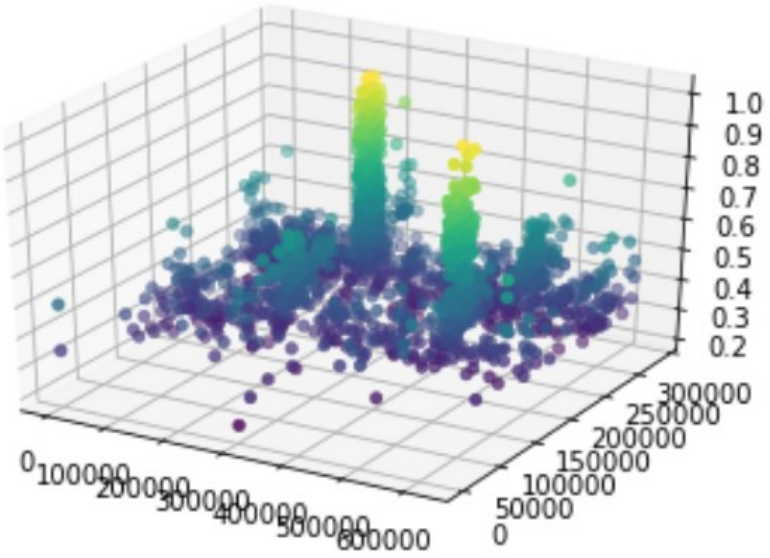


Figura 17: Grafic cu segmentele extrase după calcularea similitudinii cosinus pe embeddings

Considerăm în planul din figura de mai sus că axa verticală este reprezentată de axa *OY*, iar planul de jos este caracterizat de axele *OX* şi *OZ*, unde valoarea de pe *OY* reprezintă valoarea similitudinii cosinus şi valorile de pe *OX* şi *OZ* reprezintă valorile pozițiilor de început a segmentelor din documentul sursă şi documentul suspicios. Bulinele cu culoare deschisă (de culoare galbenă sau cu o nuanță spre galben) reprezintă segmentele între care similitudinea cosinus are o valoare mai mare (spre *1*), deci unde posibilitatea de a fi un text plagiat este mai mare.

M-am folosit de o valoare de prag pentru similitudinea cosinus, determinată experimental. Am setat acest prag ca fiind de *0.75*. Cu toate acestea, nu este suficient de a extrage doar valorile care conțin acest prag. Se poate întâmpla ca valorile alăturate, care deși au similitudine cosinus mai mică, să reprezinte secțiuni de text modificate. Luând acest fapt în considerare, am selectat și segmentele care se intersectează sau sunt la distanță de maxim 100 de caractere cu segmentele care depășesc pragul menționat sau cu alte segmente selectate.

Operația menționată este descrisă de următoarea secvență de instrucțiuni:

```
loop = True

while loop:
    loop = False

    For each candidate in segments_to_extract:
      For each valid_segment in extracted_segments:
        If candidate overlap  valid_segment and the distance between segments < 100:
          extracted_segments.append(candidate)
          loop = True
```

Unde în *segments_to_extract* se află segmentele care nu au trecut de pragul setat, iar în *extracted_segments* se află inițial segmentele care au trecut de pragul setat.

Planul 3D prezentat anterior s-ar transforma astfel:

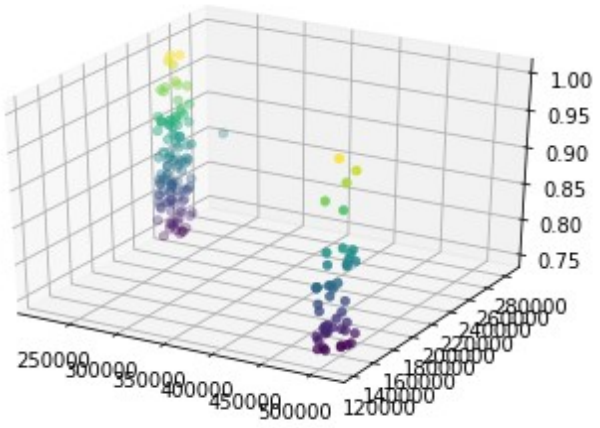


Figura 18: Grafic cu segmentele filtrate

În exemplul de mai sus, se observă că segmentele rămase formează, cel puțin vizual, două clustere. Segmentele rămase, dacă ar fi unite, ar putea forma două secțiuni continue, de dimensiune apropiată atât pentru documentul sursă, cât și pentru documentul suspicios.

Pentru a putea forma clusterele menționate, m-am folosit *DBSCAN (**D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise)*. Motivele pentru care am folosit *DBSCAN* în locul altor algoritmi de clustering sunt următoarele:

- Clusterele rezultate au o formă arbitrară, ceea ce este un punct important ținând cont că mărimea documentelor poate varia, iar asupra pasajelor plagiate pot exista diverse modificări aplicate, cum ar fi *POS - preserving Word Shuffling*.
- Numărul de clustere nu trebuie specificat. Așa cum am menționat inițial, documentele suspicioase pot avea o întindere de plagiat de până la 100%, unde pot exista un număr multiplu de pasaje plagiate din diverse documente sursă.
- Determina zonele cu densitate mare pentru a forma clustere, astfel putând să rămână segmente de text care nu aparţin niciunui cluster. Aceste segmente pot reprezenta asemănări de fraze uzuale, cum ar fi formule de politețe.

Parametrii pentru *DBSCAN* i-am setat astfel:

- *min_samples=3* – care reprezintă numărul de segmente apropiate necesare de a forma un cluster. Orice grupare mai mică de 3 segmente va fi ignorată.
- *eps=SECTION_POSITION_MAX_DIF * SECTION_POSITION_MAX_DIF / 1.5,* unde *SECTION_POSITION_MAX_DIF* are valoarea *100*, adică aceeaşi valoare maximă folosită anterior pentru filtrarea segmentelor candidat pentru clustering. Valoarea aceasta a fost determinată în mod experimental.

Obținerea clusterelor pe baza *DBSCAN* e realizată în cadrul metodei *_make_clusters* din clasa *Predictions.*

```
def _make_clusters(self):
  features = []

  for it in self._extracted_predicted_segments:
    features.append([it.this_segment.offset, it.source_segment.offset, it.similarity_value])

  X = np.array(features)
```

```
clustering_model = DBSCAN(eps=DBSCAN_EPS, min_samples=DBSCAN_MIN_SAMPLES)
clusters = clustering_model.fit(X[:, 0:2])

if SHOW_DETAILS == True:
  plot_similar_segments_3d_by_cluster(self._extracted_predicted_segments, clusters.labels_)

return X, clusters.labels_
```

Pentru segmentele rezultate din figura 18, rezultă două clustere după aplicarea DBSCAN. De asemenea, în figura de mai jos se poate observa că un segment a fost „ignorat” în procesul de clustering (fiind conturat cu altă culoare față de celelalte clustere).

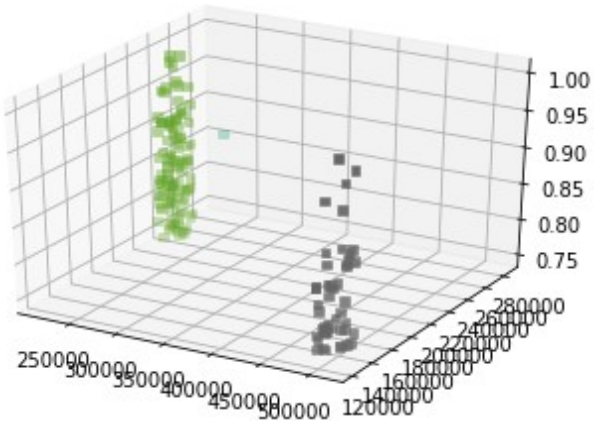


Figura 19: Grafic cu clusterele obținute după DBSCAN

Pentru unirea segmentelor dintr-un cluster, poziția de început va fi reprezentată de cea mai mică poziție de început din cluster. Dimensiunea textului din pasaj o calculez ca fiind numărul de caractere către cea mai mare poziție de final (poziția de final fiind poziția de început adunată cu dimensiunea textului). Calculele pentru poziția și dimensiunea textului se fac separat pentru documentul sursă și documentul suspicios.

Pe lângă poziția de început și poziția de final, calculez o medie progresivă a similitudinii cosinus pentru segmentele din cluster. Această medie reprezintă un mod de a selecta segmentele în cazul în care, pentru același document suspicios, se descoperă mai multe detecții care se suprapun. (păstrez doar pasajul care are cea mai mare similitudine cosinus, calculată prin media progresivă menționată)

Unirea segmentelor din clusterele determinate de *DBSCAN* se realizează în funcția *_compute_predictions* din clasa *Predictions*.

```
for seg, label in zip(self._extracted_predicted_segments, labels):
  if label != -1:
    if first_seg[label] == False:
      first_seg[label] = True
      result_seg[label] = seg
      similarity_value_count[label] = seg.similarity_value
      objects_passed[label] += 1
    else:
      result_seg[label] = self._merge_segments(result_seg[label], seg)
      similarity_value_count[label] += seg.similarity_value
      objects_passed[label] += 1
```

După care rezultatul este adăugat doar dacă nu este în conflict (nu se suprapune) cu alte predicții cu alte documente sursă anterioare. Această verificare este realizată în metoda *add_prediction*:

```
for prediction in self.plagiarism_predicted:
  conflicts = False

  for result in self.results[suspicious_doc]:
    if prediction.check_same_plagiarism_segment(result):
      conflicts = True
      break

  if not conflicts:
    if SHOW_DETAILS == True:
      print(prediction)

    self.results[self.current_suspicious_doc].append(prediction)
```

Ca exemplu, pentru clusterele din figura 19, rezultă următoarele pasaje detectate ca fiind plagiate:

Pasajul 1:

- this_segment: offset: *215971* length: *18494*

- source_segment: offset: *224618* length: *18542*

similarity_value: *0.8240924591713763*

Pasajul 2:

- this_segment: offset: *492962* length: *19303*

- source_segment: offset: *122826* length: *19649*

similarity_value: 0.8240924591713763

3.3.5 Implementarea metricilor de evaluare

Metricile de validare au fost implementate pe baza formulelor descrise în cadrul descrierii setului de date. Atât formulele pentru precizia și reapelul macro-averaged, cât și a granularității reprezintă rezultatul unor medii.

Inițial calculez sumele necesare pentru precizie, reapel și recall. Input-ul pentru aceste sume va fi reprezentat de lista de rezultate pentru fiecare document suspicios, cât și un dicționar de caracteristici extras din cadrul fișierelor XML de descriere.

Pe lângă scorul total returnat de metricile de validare, acestea mai afișează:

- numărul de detecții *true positive* – detecții care s-au suprapus cu pasajul plagiat
- numărul de detecții *false positive* – detecțiile care nu s-au suprapus cu niciun pasaj plagiat
- numărul de pasaje *false negative* – pasajele plagiate pentru care nu s-a făcut nicio detecție
- valoarea reapelului
- valoarea preciziei
- valoarea scorului F
- valoarea granularității

Numărul *true negative* nu este relevant de afișat, datorită dimensiunii mari a setului de date.

Exemplu al datelor de ieșire în urma evaluării rezultatelor:

```
Recall:  0.6838848707271423
Precision:  0.9999506147856763
Granularity:  1.025
F1:  0.8122540507191642
True positive:  32
False positive:  0
False negative:  9
Overall score:  0.7979532067614648
```

Figura 20: Exemplu afișare a rezultatelor metricilor de evaluare

În cadrul implementării pentru clasa *ValidationMetrics*, folosesc funcția *add_to_evaluate* pentru a adăuga o predicție nouă în procesul actual de calculare a scorului (pentru a adăuga noi valori în sumele necesare pentru a calcula metricile macro-averaged).

```
def add_to_evaluate(self, predictions_res, features_dict):
  for feature_key in features_dict: # features_key = the doc name
    for feature in features_dict[feature_key].plagiarism_list:
      if feature_key in predictions_res.results:
        self.recall += self.get_recall(feature.this_offset, feature.this_length, feature.source_reference,
predictions_res.results[feature_key])
        self.gran += self.get_granularity(feature.this_offset, feature.this_length, feature.source_reference,
predictions_res.results[feature_key])
      else:
        self.gran += 1

      self.n_features += 1

   for result_key in predictions_res.results: # result_key = the doc name
    for pred in predictions_res.results[result_key]:
      pred_prec = self.precision # for false/true positive
      self.precision += self.get_precision(pred.this_segment.offset, pred.this_segment.length,
pred.source_reference, features_dict[result_key].plagiarism_list)
      self.n_prec += 1

      if pred_prec != self.precision:
        self.false_positive_list.append(pred)
      else:
        self.true_positive_list.append(pred)
```

După care, pe baza sumelor calculate, calculez şi afişez rezultatele în metoda *get_overall_score*:

```
def get_overall_score(self):
  if self.n_prec == 0:
    if self.n_features == 0:
      print("No plagiarism")
      return "No plagiarism"

  if self.n_features != 0:
    self.recall /= self.n_features
    print("Recall: ", self.recall)

  if self.n_prec != 0:
    self.precision /= self.n_prec
    print("Precision: ", self.precision)
```

```
if self.n_features != 0:
  self.gran /= self.n_features
  print("Granularity: ", self.gran)
else:
  self.gran = 1

if self.precision != 0 and self.recall != 0:
  f1 = self.get_f1(self.precision, self.recall)
else:
  f1 = 0

print("F1: ", f1)

print("True positive: ", self.true_positive)
print("False positive: ", self.false_positive)
print("False negative: ", self.false_negative)

return f1 / math.log(self.gran + 1, 2)
```

3.4 Rezultate și experimente

Rezultatele și experimentele au constat în rularea pipeline-ului propus pentru subseturi de dimensiuni diverse din setul de date prezentat.

Un punct de referință inițial poate să fie reprezentat de rularea metricilor de validare peste un pipeline modificat, în care am păstrat doar *Encoplot* ca principală metodă de detectare a segmentelor plagiate, care ulterior sunt parcurse și unite printr-o metodă greedy.

Metoda de detecție greedy a constat în parcurgerea rezultatelor obținute prin Encoplot simultan pentru documentul sursă și documentul suspicios, și verificarea salturilor aproximativ egale (diferență mai mică de *10* caractere). Cerința de a forma un segment este cea de a avea minim 30 de poziții găsite care au încălcat de maxim *4* ori regula menționată sau au avut un salt mai mare de *200* de caractere.

Bineînțeles, metoda greedy nu a reprezentat potrivită sau o metodă suficientă pentru a forma segmentele finale, detaliu ce poate fi observat în figura de mai jos:

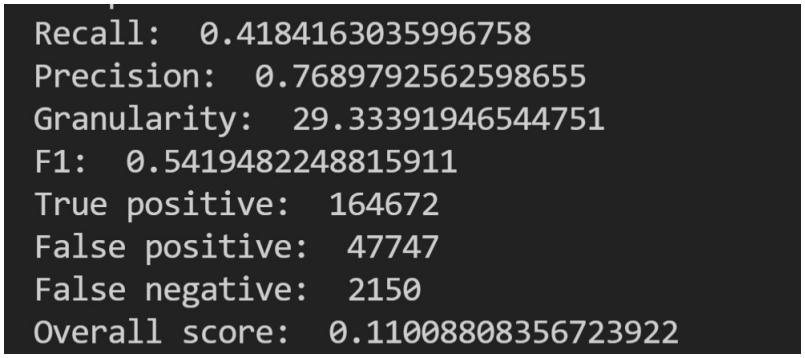


Figura 21: Rezultate Encoplot împreună cu construcţia segmentelor prin metodă greedy

Rezultatul din imaginea de mai sus a fost obținut dintr-un subset al setului de date prezentat, format din primele *1 025* documente suspicioase și *14 429* documente sursă.

Justificarea pentru rezultatele de mai sus poate fi diversă și împărțita pe mai multe puncte:

- Valoarea reapelului (aprox. 0.41) poate fi justificat de restricțiile propuse pentru a forma un segment. Cu toate acestea, dacă restricțiile ar fi fost mai mici, pentru aceiași metodă, valoarea preciziei ar fi scăzut, fiind șanse mai mari de a detecta fraze mai mici și mai comune.

- Dacă am putea privi precizia ca o metrică definitorie (având valoarea de aprox. 0.76), ar însemna că valorile de întoarse de Encoplot, împreună cu o metodă bună de a filtra și construi segmentele, constituie într-adevăr niște rezultate relevante. Într-un cadru real, o precizie de 76% pentru detectarea plagiatului, împreună cu o a doua validare umană a rezultatelor, poate reprezenta o soluție.

- Numărul disproporționat de valori *True positive (164 672*) justifică valoarea mare a granularității (aprox. 29), cauza putând fi reprezentată de metoda de selecție. Se poate interpreta că pentru o singură secțiune plagiată, s-au făcut *29* de detecții distincte.

Explicațiile de mai sus se pot concluziona astfel: Encoplot reușește să ofere informații relevante despre secțiunile plagiate, dat fiind că în orice modificare de text pot exista fragmente care să nu se fi modificat (de exemplu, rădăcinile unor cuvinte sau cuvinte preluate în plagiatul lingvistic, sau fraze care seamănă din punct de vedere al substructurii, chiar dacă cuvintele au fost schimbate). Cu toate acestea, este clar ca Encoplot nu oferă rezultate pe baza similitudinii semantice, spre deosebire de un *transformer*, care datorită conceptului de atenție, poate depista similitudinea între fraze, chiar dacă cuvintele sau frazele pot să nu aibă nicio potrivire din punct de vedere a structurii lor.

Cu toate acestea, un *transformer*, ca multe modele de învățare automată care au la bază conceptul de rețea neuronală, este foarte costisitor din punct de vedere computațional pentru a documente de mărimea unor cărți.

Astfel, pipeline-ul propus folosește inițial Encoplot care este mai puțin costisitor din punct de vedere computațional, urmat de *transformer* doar pentru pasajele formate pe baza Encoplot (conform descrierii din capitolul *PREPROCESAREA SEGMENTELOR*). Justificarea este ca există o șansă mare ca un pasaj plagiat să conțină un subset identic, de dimensiune mica (cel puțin 16 caractere), din pasajul original, iar ulterior să se poate verifica prin *transformer* și valoarea *similitudinii cosinus* dacă există și similitudine semantică.

Pentru următoarele experimente, am extras subseturi mici din documentele sursă și documentele suspicioase, în care am inclus toate documentele sursă în care existată secțiuni plagiate conform caracteristicilor din documentele XML.

Experimentele de rulare pe baza cărora am realizat experimentele conțin următorii pași:

1. Selectarea documentului suspicios pentru care se caută pasajele plagiate
2. Formarea unei liste cu documentele sursă
3. Apelarea funcției *scan_for_plagiarism* și salvarea predicțiilor returnate
4. Extragerea caracteristicilor pentru documentul curent într-un dicționar. Cheia dicționarului este reprezentat de numele documentului suspicios
5. În cazul în care se dorește rularea pentru mai multe documente suspicioase, se repetă pașii 1 – 4 pentru fiecare document suspicios, iar la final se concatenează predicțiile și se actualizează dicționarul
6. Se evaluează rezultatele predicțiilor pe baza dicționarului cu caracteristici

Pașii menționați anterior pot fi exemplificați de codul din următoarea celulă, unde se caută pasajele plagiate față de 3 documente sursă:

```
# Running example

SHOW_DETAILS = True

# the suspicious doc path to run
sus_doc_path = "/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-corpus/
suspicious-documents/part1/suspicious-document00001.txt"

# the list with the source doc files to checked
source_docs_list = ["/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-corpus/
source-documents/part1/source-document01409.txt",
            "/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-corpus/source-
documents/part1/source-document00001.txt",
            "/content/drive/MyDrive/ColabNotebooks/ColabData/external-analysis-corpus/source-
documents/part1/source-document00002.txt"]
```

```
# scan for plagiarism
pred_result = scan_for_plagiarism(sus_doc_path, source_docs_list)

# extract the features in a dictionary
features_list_dic = {}
features_list_dic[extract_path_and_doc_name(sus_doc_path)[1] + ".txt"] =
FeaturesList(sus_doc_path[:-4] + ".xml")

# evaluate the result
metrics = ValidationMetrics()
metrics.add_to_evaluate(pred_result, features_list_dic)
print("Overall score: ", metrics.get_overall_score())

SHOW_DETAILS = False
```

Pentru a verifica parametrii setați pentru *DBSCAN*, am urmărit din punct de vedere vizual, cât și ca rezultate a metricilor de validare, comparațiile dintre documentele filtrate utilizând *Encoplot* dintr-un subset foarte mic de documente (16 documente suspicioase și 17 documente sursă). Pe paginile următoare se află reprezentate grafic câteva dintre rezultatele procesului de selectare și de clustering, descrise în capitolele anterioare:

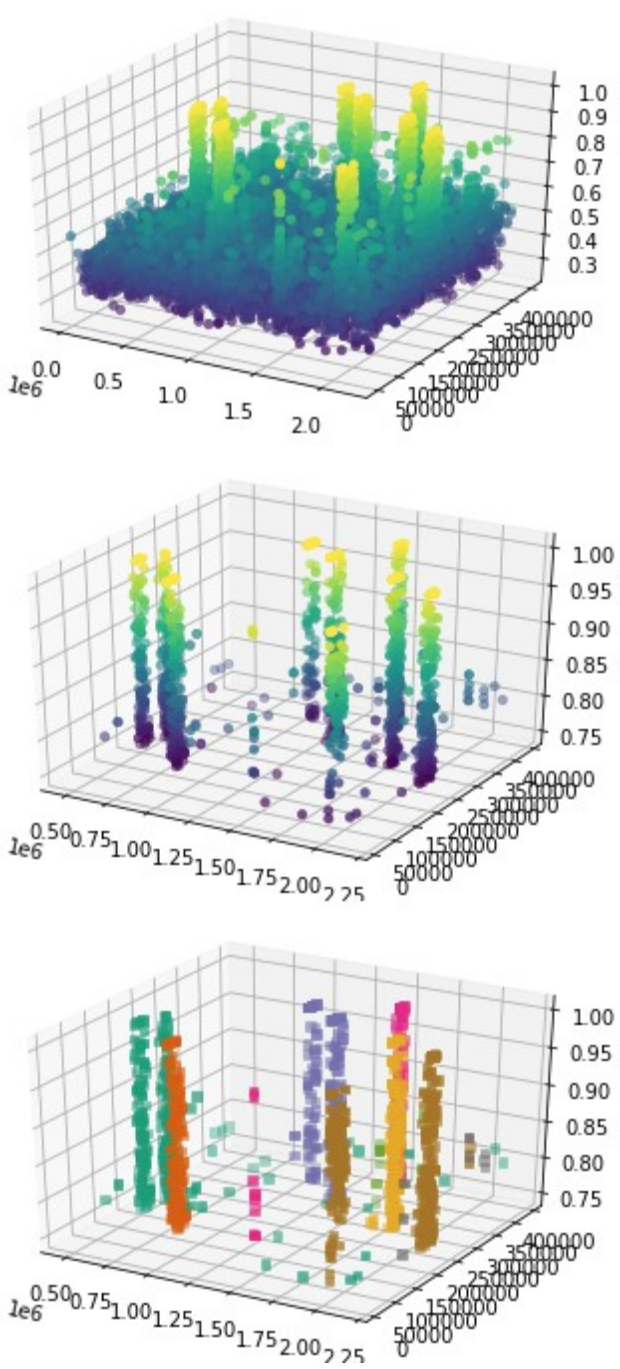


Figura 23: Exemplu cu grafice pentru paşii din pipeline

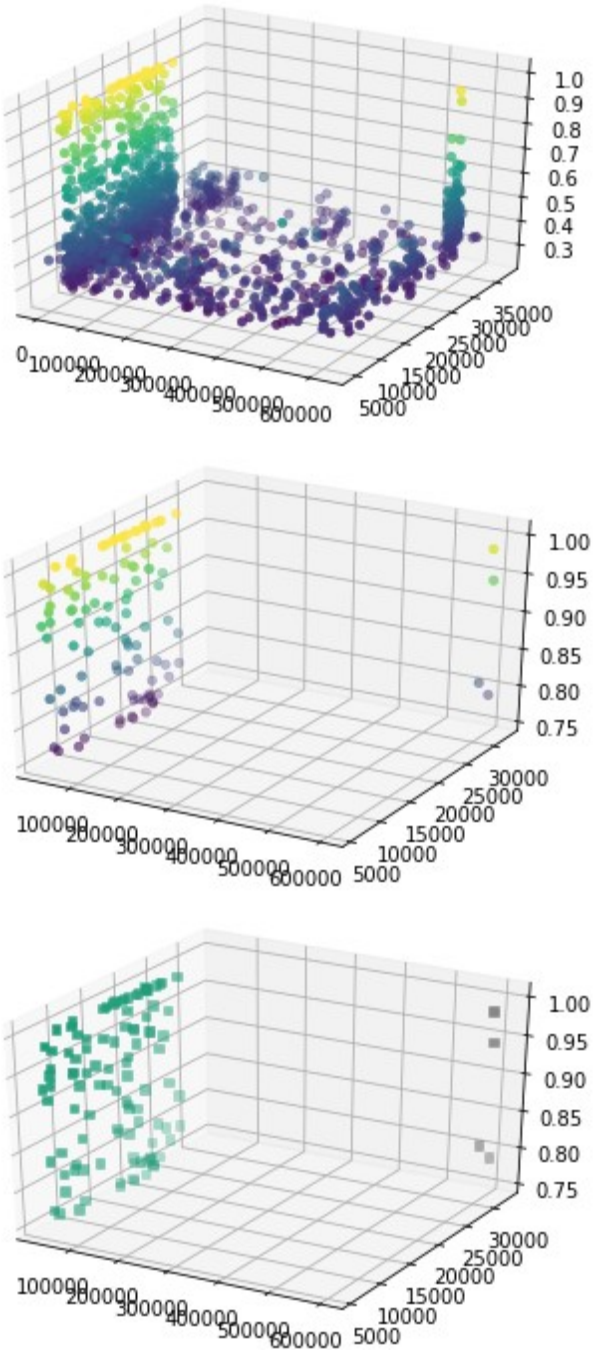


Figura 22: Exemplu cu grafice pentru paşii din pipeline

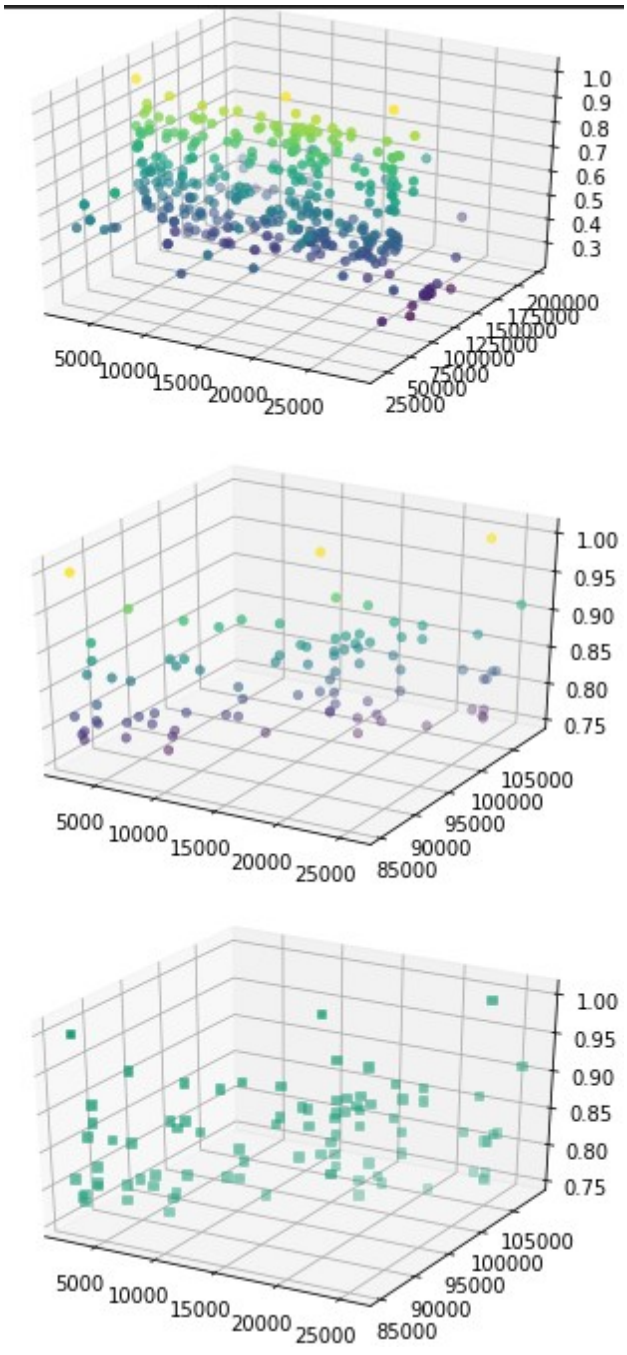


Figura 25: Exemplu cu grafice pentru pașii din pipeline

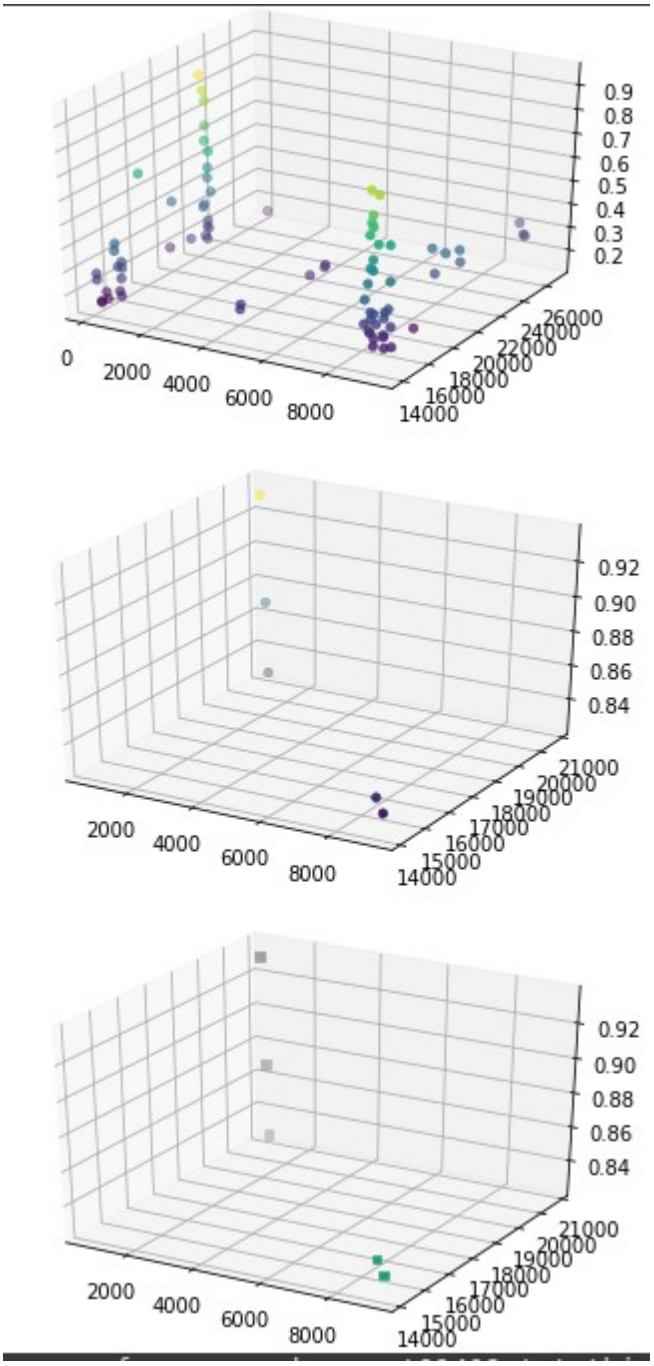


Figura 24: Exemplu cu grafice pentru pasii din pipeline

Rezultatele pentru cele 16 documente suspicioase și 17 documente sursă sunt următoarele:

```
Recall:  0.6838848707271423
Precision:  0.9999506147856763
Granularity:  1.025
F1:  0.8122540507191642
True positive:  32
False positive:  0
False negative:  9
Overall score:  0.7979532067614648
```

Figura 26: Rezultate pentru 16 documente suspicioase și 17 documente sursă

Dată fiind dimensiunea mică a subsetului, rezultatele nu sunt definitorii, însă se poate observa un punct de plecare bun. Deși s-au făcut *272* de comparații între documente (fiecare document suspicios comparat cu fiecare document sursă), nu s-a detectat niciun pasaj *false positive*, de unde rezultă precizia de *0.99*. De asemenea, algoritmul de clustering se dovedește eficient, granularitatea fiind de aproximativ *1* pentru *32* de detecții. La o analiză mai amănunțita, am observat că cele *9* detecții *false negative* reprezintă pasaje mici, care nu au fost formate deoarece nu prin metoda propusă și parametrii propuși nu s-au format clustere (nu s-a atins numărul minim de perechi pentru a forma un cluster, deci o detecție). Soluția poate fi reprezentată de modificarea parametrilor (de exemplu, micșorarea pragului necesar pentru valoarea similitudinii cosinus), sau prin adăugarea unei metode suplimentară pentru acele perechi. Cu toate acestea, acestea nu sunt singura cauză pentru valoarea reapelului mai scăzută. O altă cauză poate fi faptul că detecțiile pentru pasajele mari nu detectează întreg pasajul, fiind porțiuni neacoperite de detecție.

Experimentele următoare au constat în verificarea rezultatelor pentru subseturi din ce în ce mai mari din setul de date.

Pentru un subset de 70 documente suspicioase și *129* documente sursă, adică *9030* comparații, rezultatele arată în felul următor:

```
Recall:  0.5956378650706663
Precision:  0.9686402364355412
Granularity:  1.0117924528301887
F1:  0.7376678122598159
True positive:  292
False positive:  8
False negative:  127
Overall score:  0.731463918041757
```

Figura 27: Rezultate pentru 70 documente suspicioase și 129 documente sursă

Pentru un subset mai mare, de 100 documente suspicioase şi *174* de documente sursă, adică *17 400* de comparaţii, rezultatele arată în felul următor:

```
Recall:  0.5566333413714288
Precision:  0.9623053967600298
Granularity:  1.0090579710144927
F1:  0.7052967377436591
True positive:  357
False positive:  11
False negative:  193
Overall score:  0.7007285527985962
```

Figura 28: Rezultate pentru 100 documente suspicioase şi 174 de documente sursă

Se observă că scorul are tendinţa de a scădea cu cât subsetul de documente, şi implicit numărul de comparaţii, este mai mare. Cu toate acestea, granularitatea tinde sa îşi păstreze scorul, iar precizia are în continuare un rezultat bun. Motivele pentru valoarea mai scăzuta a reapelului sunt aceleaşi ca cele observate la punctele anterioare: pe de o parte existenţa pasajelor mici, care sunt filtrate de metodele descrise în pipeline, cât şi faptul că pasajele mari nu acoperă întreaga dimensiune a pasajului plagiat.

Pentru un subset substanţial mai mare, rezultatele arată în felul următor:

```
Recall:  0.4250861324413656
Precision:  0.6955228912155037
Granularity:  1.0055136390017412
F1:  0.5276722382377771
True positive:  1694
False positive:  695
False negative:  1771
Overall score:  0.52558473257132
```

Figura 29: Rezultate pentru 649 documente suspicioase şi 14 429 documente sursă

Subsetul cu rezultatele de mai sus are *649* documente suspicioase şi *14 429* documente sursă, adică predicţiile au constat în *9 364 421* de perechi documente comparate. Deoarece computaţional ar fi durat foarte mult pentru a verifica toate documentele sursă filtrate de *Encoplot*, am ales să verific pentru fiecare document suspicios doar perechile formate cu primele *10* documente sursă în funcţie de numărul de offset-uri dat de *Encoplot*. Acesta, împreună cu numărul mare de comparaţii, poate fi unul dintre motivele scorului mai mic. Cu toate acestea, cu o precizie de aproape 70%, pipeline-ul poate reprezenta o soluţie pentru un set de documente de comparat într-o problemă.

Într-un final, am rulat pipeline-ul pe un subset egal cu cel prezentat pentru rezultatele de la începutul capitolului (unde a fost rulat doar pentru *Encoplot*, cu segmente construite prin metoda greedy). Subsetul a fost format din primele *1 025* documente suspicioase şi *14 429* documente sursă, adică *14*

789 725 de comparații. Deși subsetul a fost mai mare decât cel anterior, rezultatele metricilor de evaluare a fost asemănător, după cum se poate observa în figura următoare:

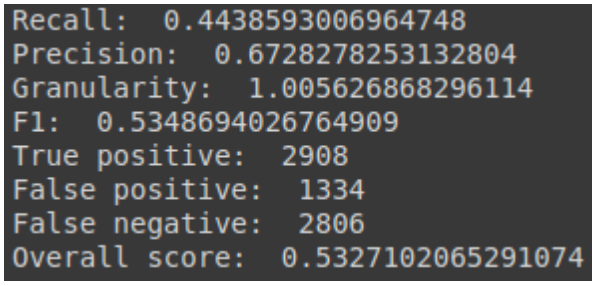


Figura 30: Rezultate pentru 1 025 documente suspicioase și 14 429 documente sursă

4 TERMENI DE UTILIZARE

4.1 Autorii

Autorii acestui document de licenţă sunt studentul Amzuloiu Andrei – Ciprian, coordonat de către coordonatorul ştiinţific, dl. Conf.dr.ing. Mihăescu Marian Cristian.

4.2 Licența de utilizare

Licența de utilizare se referă la autorii acestui proiect de licenţă. Ei au dreptul de utilizare.

5 CONCLUZII

Ca și concluzie, pipeline-ul prezentat în acest proiect, împreună cu parametrii necesari, are potențialul de a fi inclus cu succes într-un caz real. Rezultatele oferite de *transformer*, împreună *DBSCAN*, s-au dovedit o adiție bună pentru o Encoplot.

Cu toate acestea, rezultatele au arătat că încă există loc de îmbunătățiri, care pot reprezenta pașii următori. Rezultatele pot fi îmbunătățite atât prin adăugarea de metode suplimentare de a trata în pipeline segmentele rămase cu posibilitatea de a îmbunătății reapelul, cât și prin *fine-tuning* pentru valorile parametrilor prezentați de-a lungul descrierii sistemului. De asemenea, pentru un număr mare de documente pipeline-ul s-a dovedit foarte costisitor din punct de vedere computațional. O soluție ar putea fi reprezentat de paralelismul pe mai multe fire de execuție.

O altă metodă de a îmbunătății rezultatele, atât din punct de vedere al metricilor scorului, cât și a performanței timpului de execuție, poate fi reprezentată de indexarea documentelor și extragerea unui număr mai mic de documente candidat prin metode de regăsire a informației. Deși pentru setul de date actual ar fi fost dificil de realizat (documentele fiind în mare parte reprezentate de cărți de ficțiune, extrase din cadrul *Project Gutenberg*), pentru un set de documente care ar reprezenta lucrări științifice indexarea extragerea documentelor candidat metodelor de regăsire a informației ar avea sens, deoarece șansa de plagiat ar fi fost cu mult mai mare între două documente care țintesc către același subiect.

Dezvoltarea acestui proiect m-a ajutat să îmi îmbunătățesc abilitățile de cercetare, cât și abilitățile tehnice necesare pentru a proiecta, implementa și îmbunătății un proiect care conține elemente de învățare automată și procesare de limbaj natural.

6 BIBLIOGRAFIE

[DEV18] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirec-tional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018

[EST96] Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm fordiscovering clusters in large spatial databases with noise. In: Kdd. vol. 96, pp.226–231, 1996

[GRO09] Cristian Grozea, Christian Gehl și Marius Popescu - ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection, 2009

[LOC19] Lo, C.k., Simard, M.: Fully unsupervised crosslingual semantic textual similaritymetric based on bert for identifying parallel data. In: Proceedings of the 23rdConference on Computational Natural Language Learning (CoNLL). pp. 206–215, 2019

[NGU19] Nguyen, H.T., Duong, P.H., Cambria, E.: Learning short-text semantic similaritywith word embeddings and external knowledge sources. Knowledge-Based Systems182, 104842, 2019

[POT09] Martin Potthast, Benno Stein, Andreas Eiselt, Alberto Barrón-Cedeño, Paolo Rosso, Overview of the 1st International Competition on Plagiarism Detection. In: 3rd PAN Workshop.Uncovering Plagiarism, Authorship and Social Software Misuse, 2009

[STE19] Pavel Stefanovič, Olga Kurasova, Rokas Štrimaitis, The N-Grams Based Text Similarity Detection Approach Using Self-Organizing Maps and Similarity Measures, 2019

[STE09] Stein, B., Rosso, P., Stamatatos, E., Koppel, M., Agirre, E.: 3rd pan workshopon uncovering plagiarism, authorship and social software misuse. In: 25th annualconference of the spanish society for natural language processing (SEPLN). pp.1–77 (2009)

[VAJ20] Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana, Practical Natural Language Processing A Comprehensive Guide to Building Real-World NLP Systems, editura O'REILLY, 2020

[ZUB19] Zubarev, D., Sochenkov, I.: Cross-language text alignment for plagiarism detectionbased on contextual and context-free models. In: Computational Linguistics andIntellectual Technologies: Proceedings of the International Conference "Dialogue.vol. 2019 (2019)

7 REFERINȚE WEB

[COL21] Colaboratory Frequently Asked Questions. Extras pe data de 23.06.2021, disponibil on-line la adresa:

https://research.google.com/colaboratory/faq.html

[GAR21] Michael J. Garbade, What is Google Colab? 2021, disponibil on-line la adresa: https://blog.education-ecosystem.com/what-is-google-colab/

[EXP20] ExpertSystem, *What is Machine Learning? A definition.* 2020, disponibil on-line la adresa: https://www.expertsystem.com/machine-learning-definition/

[HAN12] Jiawei Han, Jian Pei, Cosine Similarity, 2012, disponibil on-line la adresa: https://www.sciencedirect.com/topics/computer-science/cosine-similarity

[POT09] Potthast, M., Stein, B., Eiselt, A., Barr´on-Cede ño, A., Rosso, P.: Pan plagiarismcorpus 2009 (pan-pc-09) , 2009. disponibil on-line la adresa: https://doi.org/10.5281/zenodo.3250083

[REI20] Reimers, N., Gurevych, I.: Making monolingual sentence embeddings multilingualusing knowledge distillation. In: Proceedings of the 2020 Conference on EmpiricalMethods in Natural Language Processing. Association for Computational Linguis-tics, 2020., disponibil on-line la adresa: https://arxiv.org/abs/2004.09813

[SAL17] Kelvin Salton do Prado, How DBSCAN works and why should we use it? 2017, disponibil on-line la adresa: https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80

[SHU18] Koo Ping Shung, Accuracy, Precision, Recall or F1? 2018, disponibil on-line la adresa: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

[SRI19] Tavish Srivastava, 11 Important Model Evaluation Metrics for Machine Learning Everyone should know. 2019, disponibil on-line la adresa: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

8 CODUL SURSĂ

Repository GIT: https://github.com/CiprianAmz/Plagiarism_Detection