



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE **CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI**



PROIECT DE DIPLOMĂ

Roșu Răzvan - Virgil

COORDONATOR ȘTIINȚIFIC

Conf. dr. ing. Mihăescu Cristian

IULIE 2020

CRAIOVA



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ

DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



Aplicație pentru detectarea plagiatului

Roșu Răzvan - Virgil

COORDONATOR ȘTIINȚIFIC

Conf. dr. ing. Marian Cristian Mihăescu

IULIE 2020

CRAIOVA

„Cel care va învăța să zboare într-o bună zi, mai întâi trebuie să învețe cum să se ridice și să meargă și să alerge și să se cațere și să danseze; nu poate învăța din zbor cum să zboare..”

Friedrich Nietzsche

DECLARAȚIE DE ORIGINALITATE

Subsemnatul *ROȘU RĂZVAN-VIRGIL*, 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 *Aplicație pentru detectarea plagiatului*,
- coordonată de Conf. dr. ing. *MARIAN – CRISTIAN MIHAESCU*,
- prezentată în sesiunea *IUNIE 2020*.

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

- 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 și 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 obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

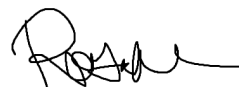
Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe 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 referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

Semnătura candidatului,

20.06.2020





UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică
Departamentul de Calculatoare și Tehnologia Informației

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

PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului/-ei:	Roșu Răzvan - Virgil
Enunțul temei:	<i>Aplicație pentru detectarea plagiatului/Implementarea unor modele de învățare automata independente pentru detectarea similarității din text, cât și compararea acestora din punct de vedere al eficienței, pentru a putea fi folosite în diferite aplicații țintă anti-plagiat.</i>
Datele de pornire:	<i>Realizarea unei implementări pentru detectarea similarității textului</i>
Conținutul proiectului:	<ol style="list-style-type: none">1. Introducere2. Informații despre domeniu și tehnologii folosite3. Design și implementare4. Concluzii5. Bibliografie6. Referințe
Material grafic obligatoriu:	<ol style="list-style-type: none">1. Documentația proiectului2. Codul sursă3. Prezentarea Power Point
Consultații:	<i>Periodice</i>
Conducătorul științific (titlul, nume și prenume, semnătura):	<i>Conf. dr. ing. MARIAN – CRISTIAN MIHAESCU</i>
Data eliberării temei:	01.12.2019
Termenul estimat de predare a proiectului:	20.06.2020
Data predării proiectului de către student și semnătura acestuia:	20.06.2020 



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică

Departamentul de **Calculatoare și Tehnologia Informației**

REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele și prenumele candidatului:

Roșu Răzvan-Virgil

Specializarea:

Calculatoare (în limba romana)

Titlul proiectului:

Aplicație pentru detectarea plagiatului

Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):

În facultate ☒

În producție ☐

În cercetare ☐

Altă locație: **[se detaliază]**

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X
Tipul proiectului		Cercetare <input type="checkbox"/>	Proiectare <input type="checkbox"/>	Realizare practică X	Altul [se detaliază]
Aparatul matematic utilizat		Simplu <input type="checkbox"/>	Mediu <input type="checkbox"/>	Complex X	Absent <input type="checkbox"/>
Utilitate		Contract de cercetare <input type="checkbox"/>	Cercetare internă <input type="checkbox"/>	Utilare X	Altul [se detaliază]
Redactarea lucrării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X
Partea grafică, desene		Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
Realizarea practică	Contribuția autorului	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Mare <input type="checkbox"/>	Foarte mare X
	Complexitatea temei	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă X
	Analiza cerințelor	Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X

	Arhitectura	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă X
	Întocmirea specificațiilor funcționale	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
	Implementarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
	Testarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
	Funcționarea	Da X	Parțială <input type="checkbox"/>	Nu <input type="checkbox"/>	
Rezultate experimentale		Experiment propriu X		Preluare din bibliografie <input type="checkbox"/>	
Bibliografie		Cărți x	Reviste x	Articole x	Referințe web x
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI X	RESPINGEREA PROIECTULUI <input type="checkbox"/>
-----------------------------------	---

Data,

20.06.2020

Semnătura conducătorului științific,

conf. dr. ing. Cristian Mihaescu

REZUMATUL PROIECTULUI

Proiectul are ca scop dezvoltarea unor diferite modele de machine learning, și cercetarea eficienței acestora în cazul detectării similarității în text respectiv a plagiatului. Toate aceste modele sunt bazate pe un dataset comun de validare, pentru a pune în lumina diferența dintre acestea în cadrul scopului proiectului. Aceste modele vor fi comparate între ele pentru a oferi o retrospectivă din punctul de vedere al performanței. Bineînțeles, aceste modele sunt independente unul față de celălalt, iar implementările sunt realizate folosind diferite metode pentru procesarea limbajului natural. Textul va trece prin o etapă de pre-procesare, în care acesta va fi convertit în vectori, stagiul în care acesta va fi în o stare optimă pentru integrarea în modele. De acolo, vor intervine clasicele metode de calculare a similarității textului folosind metoda similarității cosinus între doi vectori, unde vectorii noștri vor fi chiar textele convertite în urma pre-procesării.

Procesarea limbajului natural este una dintre componentele cheie din Inteligența Artificială, care oferă abilitatea de a face o mașină să recunoască limbajul uman. Aceasta tehnica ajută mașinile să înțeleagă și să extragă diferite pasaje din o dată de tip text prin aplicarea diferitelor tehnici cum ar fi, similaritatea textului, regăsirea informației, clasificarea documentelor, extragerea entităților și în un final, clustering. Similaritatea text este una dintre tehnicile esențiale de NLP care este folosită pentru a găsi cât de apropiate sunt două bucăți de text atât din punct de vedere sintactic, cât și semantic.

Modelele sunt implementate după o metodologie clasică:

1. Pre-procesarea textului
2. Extragerea caracteristicilor
3. Similaritatea vectorilor obținuți

Pentru ca soluția să fie oferită în un mod optim, am parcurs câteva cursuri de început și tutoriale legate atât de învățarea automată, cât și de procesarea limbajului natural. Unele metode pe care am dorit inițial să le implementez (cum ar fi k-means) în urma studiului s-au dovedit a fi ineficiente, drept pentru care implementarea acestora nu mi s-a părut necesară. Aceste metode au fost de asemenea implementate în limbajul Python, la fel ca și restul modelelor.

Termenii cheie: BERT, Transformer, GloVe, TF-IDF, Machine Learning, Python, Word Embeddings, Similaritate Cosinus.

MULȚUMIRI

Mulțumesc pe aceasta cale conducătorului științific, domnul Conf. Dr. Ing. Marian Cristian Mihăescu, care m-a îndrumat pe toată durata realizării proiectului și care mi-a oferit sprijin informațional pentru a putea realiza cu succes acest proiect, cât și tuturor cadrelor didactice din aceasta facultate pentru toate cunoștințele acumulate în de-a lungul anilor, cât și pentru faptul că și-au pus amprenta în dezvoltarea mea personală.

De asemenea, aș dori să le mulțumesc tuturor persoanelor care m-au susținut și călăuzit pe parcursul elaborării lucrării de licență, de la profesori pentru îndrumare până la familie pentru suportul moral și sfaturile oferite.

CUPRINSUL

1	INTRODUCERE.....	1
1.1	SCOPUL.....	1
1.2	MOTIVAȚIA	1
1.3	OBIECTIVE.....	1
2	INFORMAȚII DESPRE DOMENIU ȘI TEHNLOGII FOLOSITE	2
2.1	ÎNVĂȚARE AUTOMATA	2
2.2	METRICI DE EVALUARE	4
2.2.1	<i>Acuratețea</i>	5
2.2.2	<i>Precizia</i>	5
2.2.3	<i>Reamintirea</i>	6
2.2.4	<i>Scorul F1</i>	6
2.3	PROCESAREA LIMBAJULUI NATURAL	6
2.4	PYTHON	7
2.5	DISTRIBUȚIA FOLOSITA	9
2.6	IDE.....	10
2.7	LIBRARII FOLOSITE.....	10
2.7.1	<i>Scikit-learn</i>	10
2.7.2	<i>Numpy</i>	10
2.7.3	<i>Tensorflow</i>	11
2.7.4	<i>Sentence Transformers</i>	12
2.7.5	<i>Kaagle</i>	12
2.8	TRANSFORMER	12
2.8.1	<i>BERT</i>	15
2.9	GLOVE	20
2.10	TF-IDF	22
2.11	SIMILARITATEA COSINUS	25
2.12	ROBERTA.....	28
3	DESIGN SI IMPLEMENTARE	30
3.1	DESCRIEREA DATASET-ULUI	30
3.2	ANTRENARE ȘI REZULTATE.....	32
3.2.1	<i>WordEmbeddings folosind GloVe + tf/idf</i>	32
3.2.2	<i>BERT</i>	36

3.2.3	<i>RoBERTa</i>	39
4	TERMENI DE UTILIZARE.....	44
4.1	AUTORII.....	44
5	CONCLUZII	45
6	BIBLIOGRAFIE.....	46
7	REFERINȚE WEB.....	47
8	CODUL SURSĂ	48
9	CD / DVD	57

LISTA FIGURILOR

FIGURĂ 1 MATRICEA DE CONFUZIE.....	5
FIGURĂ 2 ARHITECTURA UNUI TRANSFORMER	13
FIGURĂ 3 SCALED ATTENTION VS MUTLI HEAD ATTENTION	14
FIGURĂ 4 STRUCTURA BERT	16
FIGURĂ 5 BERT EMBEDDINGS	18
FIGURĂ 6 MASKED LM VS LEFT TO RIGHT	20
FIGURĂ 7 FORMULA SIMILARITĂȚII COSINUS	26
FIGURĂ 8 SIMILARITATEA COSINUS	26
FIGURĂ 9 CELE 3 DOCUMENTE EXEMPLU	27
FIGURĂ 10 PROIECTAREA DOCUMENTELOR	28
FIGURĂ 11 DETALII DATASET STSBENCHMARK.....	31
FIGURĂ 12 GRAFIC ACURATEȚE MODEL.....	32
FIGURĂ 13 MODEL ANTRENARE GLOVE-TFIDF	32
FIGURĂ 14 TIMP/EPOCH	35
FIGURĂ 15 REZULTATE DUPA EPOCH 1	35
FIGURĂ 16 EXEMPLU DE VERIFICARE GLOVE+ TF-IDF.....	35
FIGURĂ 17 REZULTATE FINALE GLOVE+ TFIDF	35
FIGURĂ 18 BERT PRE-ANTRENAT INTEGRAT IN APLICAȚIE.....	36
FIGURĂ 19 BERT ANTRENAT DUPĂ EPOCH 0	37
FIGURĂ 20 REZULTAT EVALUARE BERT	37
FIGURĂ 21 REZULTAT BERT DUPA CELE 4 EPOCHS.....	37
FIGURĂ 22 EXEMPLU VERIFICARE BERT	38
FIGURĂ 23 ROBERTA CONFIGURATION	39
FIGURĂ 24 ROBERTA ANTRENAT DUPA EPOCH 0	39
FIGURĂ 25 REZULTAT ROBERTA.....	40
FIGURĂ 26 EXEMPLU VERIFICARE ROBERTA.....	40
FIGURĂ 27 PIPELINE VERIFICARE MODEL	40
FIGURĂ 28 PIPELINE ANTRENARE MODEL + EVALUARE	41

LISTA TABELELOR

TABEL 2: VARIANTE DE PONDERE A FRECVENȚEI TERMENILOR(TF).....	23
TABEL 3: VARIANTE DE PONDERE A IDF IN DOCUMENTE	24
TABEL 4: SCHEME DE PONDERARE RECOMANDATE PENTRU TF-IDF	25

1 INTRODUCERE

1.1 Scopul

Scopul realizării acestui proiect este acela de cerceta și verifica folosirea anumitor modele de învățare automata pentru detectarea plagiarismului din un text scris. Aceste modele pot fi ulterior implementate în diferite aplicații de detectare a plagiatului (pagini web, aplicații mobile, aplicații desktop) .

Pentru a putea testa rezultatul final al fiecărui model folosind date cât mai apropiate de realitate, totalitatea datelor disponibile a fost împărțită în procente pentru validarea și testarea rezultatului.

1.2 Motivația

Motivația vine din dorința mea de a cerceta și de a găsi modul cel mai bun de detectare a plagiatului, folosind învățarea automata și de a acumula mai multă experiență în dezvoltarea produselor software bazate pe platforma Python. De asemenea, ideea în sine de plagiarism, care mi s-a părut foarte interesantă, m-a motivat să o fac în cadrul proiectului meu de diploma.

1.3 Obiective

O1. Dezvoltarea unui mod de pre-procesare al datelor necesare pentru antrenarea modelelor.

Acest obiectiv al lucrării de disertație vizează dezvoltarea unui modul care să pregătească textul folosit pentru detectare. De aceea, pentru ca textul să fie apoi transformat în vectori, sunt necesare câteva funcții de pre-procesare a lui, în care, vor fi șterse toate caractere non-ASCII, caracterele speciale, ș.a.m.d . Pentru a putea realiza acest obiectiv este necesară implementarea unui fișier care să facă aceasta pre-procesare.

O2. Dezvoltarea unui mod de extragere a caracteristicilor

Acest obiectiv al lucrării de disertație vizează dezvoltarea unui modul care să prelucreze textul folosit pentru detectare. Textul va fi transformat în vectori, deoarece modelele noastre lucrează folosind date de intrare vectori. Textul va fi parcurs și clusterizat astfel încât modelul să poată înțelege textul. Aceasta parte poate fi vizualizată ca o encodare a textului în un format numeric, recunoscut de model. Textul căutat va fi transformat folosind o metoda de extragere în un format numeric, ca un vector, și acesta va fi comparat cu ceva pre-antrenat.

O3. Dezvoltarea unui mod de realizare a similarității între cei 2 vectori

Acest obiectiv al lucrării de disertație vizează dezvoltarea unui mod pentru a calcula similaritatea dintre cei 2 vectori rezultați la punctul anterior. Deoarece word-embeddings-urile generate trebuie să fie comparate pentru a obține o similaritate semantică între cei 2 vectori, este necesară implementarea unei distanțe care ar putea calcula această similaritate prin o valoare în intervalul $[0,1]$, unde 0 – similaritate foarte scăzută, 1 – identice. Aceasta distanță va fi reprezentată prin similaritatea cosinus, una dintre cea mai folosită metoda pentru compararea a 2 vectori. Aceasta similaritate este producția unui punct între 2 vectori. Dacă unghiul are o valoare de 0 radiani, aceasta similaritate va fi 1.

2 INFORMAȚII DESPRE DOMENIU ȘI TEHNologii FOLOSITE

Pentru a putea oferi un livrabil în cadrul proiectului de diploma, diferite domenii, concepte, limbaje de programare și librării pentru acestea au fost studiate pe parcursul dezvoltării produsului, unele dintre acestea fiind chiar aplicate pentru prima dată.

2.1 Învățare automată

„*Învățarea automată* (în engleză, „Machine Learning”) este un subdomeniu al informaticii și o ramură a *inteligenței artificiale*, al cărui obiectiv este de a dezvolta tehnici care dau calculatoarelor posibilitatea de a învăța. Mai precis, se urmărește să se creeze programe capabile de generalizare pe baza unor exemple.

Este, prin urmare, un proces inductiv. În multe cazuri, domeniul învățării automate se suprapune cu cel al statisticii computaționale, deoarece cele două discipline se bazează pe analiza datelor. Cu toate acestea, învățare automată, se concentrează și pe complexitatea computațională al problemelor. Multe probleme sunt în clasa „NP-hard” așa că o mare parte din cercetările efectuate asupra procesului de învățare automată sunt axate pe proiectarea de soluții viabile la aceste probleme. Învățare automată poate fi văzut ca o încercare de a automatiza unele părți din *metoda științifică*, folosind metode matematice.

Învățarea automată are o gamă largă de aplicații, inclusiv **motoarele de căutare**, diagnostice medicale, detectare de fraudă în utilizarea **cardului de credit**, analiză a pieței de valori, clasificarea a secvențelor de ADN, recunoaștere a vorbirii și **limbajului scris**, jocuri și **robotică**.

Unele sisteme de învățare automată încearcă să elimine toată nevoia de intuiție sau cunoștințe de specialitate din procesele de analiză a datelor, în timp ce alții încearcă să stabilească un cadru de colaborare între expert și computer. Cu toate acestea, intuiția umană nu poate fi înlocuit în totalitate, deoarece proiectantul sistemului trebuie să se precizeze forma de reprezentare a datelor și metodele de manipulare și caracterizare a acestora. Cu toate acestea, calculatoarele sunt utilizate în întreaga lume în scopuri tehnologice foarte bune.

În învățare automată putem obține 3 tipuri de cunoaștere, care sunt:

- **Creșterea**, cunoașterea care este dobândită din ceea ce ne înconjoară, care păstrează informațiile în memorie ca și cum ar fi lăsat urme.
- **Restructurarea**, cunoașterea care este folosită pentru a interpreta cunoștințele individuale și care generează noi cunoștințe.
- **Ajustarea**, obținută prin generalizarea mai multor concepte sau prin generarea unor concepte proprii.

Cele trei tipuri se desfășoară în timpul unui proces automat de învățare, dar importanța fiecărui tip de cunoaștere depinde de caracteristicile a ceea ce se încearcă să învețe. Învățarea este mai mult decât o necesitate, este un factor prioritar pentru a satisface nevoile de inteligență artificială.

Învățarea automată are ca rezultat un model pentru a rezolva o anumită sarcină. Printre modele se disting:

- **Modele geometrice**, construite în spațiul de instanțe și care pot avea unul, două sau mai multe dimensiuni. Dacă există o margine de decizie liniară între clase, se spune că datele sunt liniar separabile. O limită de decizie liniară este definită ca $w \cdot x = t$, unde w este un vector perpendicular pe limita de decizie, x este un punct arbitrar de pe granița de decizie, iar t este pragul de decizie.
- **Modele probabilistice**, care încercă să determine distribuția între probabilitate de descripție pentru funcția care leagă valorile de caracteristici cu valorile determinate. Unul dintre conceptele cheie pentru dezvoltarea modelelor probabilistice este statistica bayesiană.
- **Modele logice**, care transformă și exprimă probabilitățile în norme organizate sub formă de arbori de decizie.”^[1]

De asemenea, algoritmi folosiți în învățarea automată mai sunt în general împărțiți în supervizați respectiv nesupervizați.

- Algoritmii de învățare automată supervizați pot aplica ceea ce a fost învățat în trecut la date noi folosind exemple etichetate pentru a prezice evenimente viitoare. Pornind de la analiza unui set de date de instruire cunoscut, algoritmul de învățare produce o funcție dedusă pentru a face predicții despre valorile de ieșire. Sistemul este capabil să ofere ținte pentru orice input nou după o pregătire suficientă. Algoritmul de învățare își poate compara, de asemenea, ieșirea cu ieșirea corectă, prevăzută și poate găsi erori pentru a modifica modelul în consecință.
- În schimb, algoritmi nesupervizați de învățare sunt folosiți atunci când informațiile utilizate pentru instruire nu sunt nici clasificate, nici etichetate. Studiul de învățare nesupervizat studiază modul în care sistemele pot deduce o funcție pentru a descrie o structură ascunsă din date neetichetate. Sistemul nu își dă seama de ieșirea potrivită, dar explorează datele și poate atrage inferențe din seturi de date pentru a descrie structuri ascunse din date fără marcă.
- Algoritmii de învățare automată semi-supravegheată se încadrează undeva între învățarea supravegheată și nesupravegheată, deoarece folosesc atât date etichetate, cât și date neetichetate, pentru formare - de obicei o cantitate mică de date etichetate și o cantitate mare de date fără marcat. Sistemele care utilizează această metodă sunt capabile să îmbunătățească considerabil acuratețea învățării. De obicei, învățarea semi-supravegheată este aleasă atunci când datele etichetate dobândite necesită resurse calificate și relevante pentru a-l instrui / învăța din aceasta. În caz contrar, achiziționarea de date fără marcă, în general, nu necesită resurse suplimentare.

2.2 Metrici de evaluare

„Acestea sunt necesare când trebuie să evaluăm cât de bine se comportă modelul nostru. Unele metrici folosite de obicei sunt acuratețea, precizia, rechemarea și scorul F1.

Actual Class	Predicted class	
	Class = Yes	Class = No
Class = Yes	True Positive	False Negative
Class = No	False Positive	True Negative

Figura 1 Matricea de confuzie

True positive și true negative sunt observațiile care sunt prezise corect și, prin urmare, sunt afișate în verde. Vrem să minimalizăm false positives și false negatives, astfel încât acestea să fie afișate în roșu.

True Positives (TP) - Acestea sunt valorile pozitive prezise corect, ceea ce înseamnă că valoarea clasei reale este da și valoarea clasei prezise este, de asemenea, da.

True Negatives (TN) - Acestea sunt valorile negative prezise corect, ceea ce înseamnă că valoarea clasei reale este nu și valoarea clasei prezise este nu.

False Positives (FP) - Când clasa actuală este nu și clasa prezisă este da.

False Negatives (FN) - Când clasa actuală este da și clasa prezisă este nu.

2.2.1 Acuratețea

Acuratețea este cea mai intuitivă măsură a performanței și este pur și simplu un raport dintre observația corect prevăzută și observațiile totale. Se poate crede că, dacă avem o acuratețe ridicată, atunci modelul nostru este cel mai bun. Da, acuratețea este o măsură excelentă, dar numai atunci când aveți seturi de date simetrice în care valorile falselor pozitive și negative negative sunt aproape aceleași. Prin urmare, trebuie să analizați alți parametri pentru a evalua performanța modelului dvs.

$$\text{Acuratețea} = \frac{TP+TN}{TP+FP+FN+TN}$$

2.2.2 Precizia

Precizia este raportul dintre observațiile pozitive prezise corect și totalul observațiilor pozitive prezise. Întrebarea conform căreia acest răspuns metric este al tuturor pasagerilor care au fost etichetați ca supraviețuit, câți au supraviețuit de fapt? Precizia ridicată se referă la rata scăzută fals pozitivă.

$$\text{Precizia} = \frac{TP}{TP+FP}$$

2.2.3 Reamintirea

Reamintirea este raportul dintre observațiile pozitive prezise corect și toate observațiile din clasa reală - da. Răspunsurile la reamintirea întrebării sunt: dintre toți pasagerii care au supraviețuit cu adevărat, câți am etichetat?

$$\text{Reamintirea} = TP/TP+FN$$

2.2.4 Scorul F1

Scorul F1 este media ponderată a preciziei și a rechemării. Prin urmare, acest scor ia în considerare atât falsele pozitive, cât și falsele negative. Intuitiv nu este la fel de ușor de înțeles ca acuratețe, dar F1 este de obicei mai util decât precizia, mai ales dacă aveți o distribuție inegală a claselor. Precizia funcționează cel mai bine dacă falsele pozitive și falsele negative ce au costuri similare. Dacă costurile falselor pozitive și falselor negative sunt foarte diferite, este mai bine să ne uităm atât la precizie, cât și la reamintire.

$$F1\ Score = 2 * (\text{Reamintirea} * \text{Precizia}) / (\text{Reamintirea} + \text{Precizia})^{[12]}$$

2.3 Procesarea limbajului natural

“*Procesarea limbajului natural* (PLN - engleză – NLP) reprezintă o tehnologie (adică un ansamblu de procese, metode, operații) care creează și implementează modalități de a executa diferite sarcini referitoare la limbajul natural (cum ar fi construcția unor interfețe-bazate pe limbaj natural-cu baze de date, traducerea automată ș.a.m.d.). Procesarea limbajului natural reprezintă și astăzi o problemă dificilă și în cea mai mare parte nerezolvată. Găsirea unei tehnologii adecvate este extrem de grea datorită naturii multidisciplinare a problemei, fiind implicate următoarele științe și domenii: lingvistică, psiholingvistică, lingvistică computațională, filozofie, informatică, în general, și inteligența artificială, în mod special, etc.

Ingineria limbajului natural se ocupă de implementarea unor sisteme de mare anvergură. Aplicațiile procesării limbajului natural se înscriu în trei mari categorii:

Aplicațiile bazate pe text, din care reamintim:

- clasificarea documentelor (găsirea documentelor legate de anumite subiecte).
- regăsirea informației (căutarea unor cuvinte cheie sau concepte).
- extragerea informației (legate de un anumit subiect, deci de un anumit cuvânt cheie).
- înțelegerea textelor (care presupune o analiză profundă a structurii acestora).

- traducerea automata și traducerea asistata de calculator dintr-o limbă în alta.
- alcătuirea sintezelor.
- achiziția cunoștințelor.

Aplicațiile bazate pe dialog, care implica comunicarea între om și mașină, aplicații cum ar fi sistemele de învățare, sistemele de interogare și răspuns la întrebări, rezolvarea problemelor, controlul (bazat pe limba vorbită) al unui calculator, ș.a.m.d.

Aplicații bazate pe procesarea vorbirii, (Este important să facem distincția între problemele de recunoaștere a vorbirii și cele de înțelegere a limbajului. Astfel, trebuie să remarcăm încă de la început faptul că un sistem de recunoaștere a vorbirii nu folosește nici un element de înțelegere a limbajului. Recunoașterea vorbirii se ocupă numai de identificarea cuvintelor vorbite provenind de la un semnal dat, nu și de înțelegerea mesajului, adică a modului în care aceste cuvinte sunt folosite în procesul de comunicare. Pentru a deveni un sistem de înțelegere a limbajului, un dispozitiv de recunoaștere a vorbirii trebuie să furnizeze intrarea să unui sistem de înțelegere a limbajului natural, operație care produce un așa-numit "sistem de înțelegere a limbajului vorbit". Caracteristică de bază a oricărui sistem de înțelegere este aceea că el realizează o reprezentare a înțelesului propozițiilor într-un limbaj de reprezentare, care poate fi utilizat în vederea unei procesări ulterioare).”^[2]

2.4 Python

“*Python* este un limbaj de programare dinamic, de nivel înalt, ce pune accent pe expresivitatea și înțelegerea ușoară a codului. Sintaxa sa permite implementări echivalente cu alte limbaje în mai puține linii de cod. Datorită acestui fapt, Python este foarte răspândit atât în programarea de aplicații, cât și în zona de scripting.

Limbajul facilitează mai multe paradigme de programare, în special paradigma imperativa (C) și pe cea orientată pe obiecte (Java). Spre deosebire de C, Python nu este un limbaj compilat, ci interpretat. Acest fapt are atât avantaje, cât și dezavantaje. Pe de-o parte, Python este mai lent decât C. Pe de altă parte, aplicațiile Python sunt foarte ușor de depanat, codul putând fi ușor inspectat în timpul rulării. De asemenea, este foarte ușor de experimentat cu mici fragmente de cod folosind interpretorul Python. Python are de asemenea un garbage collector.

Sintaxa este gândită în așa fel încât programele Python să fie ușor de citit. Acest lucru este obținut prin folosirea de cuvinte în locul semnelor (de exemplu, and în loc de &&) și prin includerea indentării în limbaj. Astfel, în Python nu se folosesc acolade (ca în C/C++, Java), ci blocurile de cod se delimitează prin indentare. Programele Python sunt, de multe ori, foarte aproape de o “implementare” echivalentă în pseudocod.

Limbajul Python este interpretat, nu compilat. Asta înseamnă că programele Python sunt transformate într-un limbaj intermediar. Acest lucru permite codului să fie ușor de portat pe diverse sisteme de operare și arhitecturi hardware.

Codul este executat linie cu linie. Astfel, dacă - de exemplu - apelăm o funcție care nu există, vom primi un mesaj de eroare abia când se încearcă executarea liniei respective. Erorile de sintaxă sunt raportate însă înainte de rularea programului.

Includerea tuturor acestor structuri, precum și a funcțiilor ce permit manipularea și prelucrarea lor, precum și multe alte biblioteci de funcții sunt prezente datorită conceptului “baterii incluse”, ce poate fi explicat prin faptul că Guido van Rossum și comunitatea ce s-a format în jurul limbajului cred că un limbaj de programare nu prezintă utilitate practică dacă nu are un set de biblioteci importante pentru majoritatea dezvoltatorilor.

Din acest motiv Python include biblioteci pentru lucrul cu fișiere, arhive, fișiere XML și un set de biblioteci pentru lucrul cu rețeaua și principalele protocoale de comunicare pe internet (HTTP, Telnet, FTP). Un număr mare de platforme Web sunt construite cu Python. Abilitățile limbajului ca limbaj pentru programarea CGI sunt în afara oricăror dubii. De exemplu [YouTube](https://www.youtube.com/), unul din site-urile cu cea mai amplă cantitate de trafic din lume, este construit pe baza limbajului Python.

Totuși, Python permite extinderea funcționalității prin pachete adiționale programate de terți care sunt axate pe o anumită funcționalitate. De pildă, pachetul *wxPython* conține metodele și structurile necesare creării unei interfețe grafice.

Popularitatea limbajului este în creștere începând cu anul [2000](#), datorită faptului că Python permite crearea mai rapidă a aplicațiilor care nu cer viteze înalte de procesare a datelor. De asemenea este util ca limbaj de scripting, utilizat în cadrul aplicațiilor scrise în alte limbaje. Modulele (bibliotecile) Python pot fi de asemenea scrise în C, compilate și importate în Python pentru a mări viteza de procesare.”^[3]

De ce Python?

Există multe motive - am enumerat unele dintre ele deja, dar să le enumerăm din nou într-o manieră mai practică:

- este **ușor de învățat** - timpul necesar pentru a învăța Python este mai scurt decât pentru multe alte limbi; asta înseamnă că este posibil să pornești mai rapid programarea reală;

- este **ușor de predat** - volumul de muncă didactic este mai mic decât cel necesar în alte limbi; aceasta înseamnă că profesorul poate pune mai mult accent pe tehnicile de programare generale (independente de limbaj), nu risipind energie pe trucuri exotice, excepții ciudate și reguli de neînțeles;
- este **ușor de utilizat** pentru scrierea de software nou - este adesea posibil să scrieți cod mai rapid când utilizați Python;
- este **ușor de înțeles** - este, de asemenea, de multe ori mai ușor de înțeles mai rapid codul altcuiva dacă este scris în Python;
- este **ușor de obținut**, instalat și implementat - Python este gratuit, deschis și multiplatform; nu toate limbile se pot lăuda cu asta.

Este utilizat pe scară largă pentru a implementa servicii complexe de internet, precum motoarele de căutare, stocarea în cloud și instrumente, social media și așa mai departe. Ori de câte ori utilizați oricare dintre aceste servicii, sunteți de fapt foarte aproape de Python, deși nu l-ați ști.

Multe instrumente de dezvoltare sunt implementate în Python. Din ce în ce mai multe aplicații de utilizare de zi cu zi sunt scrise în Python. Mulți oameni de știință au abandonat instrumentele scumpe și au trecut la Python.

Mulți testerii de proiecte IT au început să utilizeze Python pentru a efectua proceduri de testare repetabile. Lista poate continua.

Fiind unul dintre cele mai populare limbaje de programare din ultima vreme și unul dintre cele mai evaluate limbaje de programare pentru învățare automată datorită multitudinii de framework-uri și librării pentru data science, acesta este în general prima opțiune când vine vorba de alegerea mediului de dezvoltare pentru un proiect de învățare automată.

2.5 Distribuția folosită

Inițial, s-a încercat distribuția Anaconda pentru Windows.

Anaconda este o distribuție de Python și R cu un management al pachetelor cât și al deployment-ului foarte simplificată. Acest lucru este un avantaj, deoarece putem folosi mai multe medii de dezvoltare în Python cu diferite configurații prin care putem cicla. Aceasta vine și cu un manager de conținut, din

care putem instala foarte ușor librării de Python, cât și să rezolvăm conflicte între diferite versiuni de librării.

Datorită faptului că unele librării nu au fost disponibile pe sistemul de operare Windows, am decis ca aplicația să fie hostată într-un virtual environment pe un subsystem de Linux, folosind un command line de Ubuntu, alături de pachetul de instalare *PIP*.

2.6 IDE

Visual Studio Code este un text-editor foarte vast care permite utilizatorului interacțiune cu o multitudine de limbaje de programare (C#, PHP, Python) precum și runtime-uri (.NET, Unity). De asemenea, Visual Studio Code are inclusiv ustensilele necesare folosind terminalul respectiv câteva plugin-uri pentru a se putea face debugging pe soluțiile adăugate în acesta. Acesta este disponibil pentru Windows, macOS și Linux. Visual Studio Code vine de asemenea cu un manager de conținut, în care putem instala diferite plugin-uri pentru a face experiența dezvoltării de produse software în aceasta ca fiind una cât mai bună.

2.7 Librării folosite

2.7.1 Scikit-learn

Scikit-learn reprezintă o librărie open source de învățare automată concepută pentru limbajul de programare Python ce oferă unelte pentru preprocesarea datelor, selectarea modelelor disponibile și evaluarea acestora.

2.7.2 Numpy

“*NumPy* este o librărie ce adaugă limbajului Python suport pentru matricele și matricele multidimensionale mari, împreună cu o colecție vastă de funcții matematice care funcționează pe aceste tablouri menționate precedent.”^[4] Implementările acestor structuri sunt făcute pentru acești vectori și matrice de dimensiuni foarte mari, recunoscute sub numele de big data.

Librăria de asemenea conține:

- o matrice obiect de dimensiune N pentru operații
- funcții clasice de algebra liniară
- transformări Fourier clasice
- integrarea de cod scris în C/C++ sau Fortran”^[4]

2.7.3 Tensorflow

Tensorflow este o bibliotecă de software gratuit, open-source pentru programarea unor fluxuri de date și funcții derivabile într-o serie de sarcini. Această bibliotecă este una matematică simbolică și este, de asemenea, utilizată pentru aplicații de învățare automată, spre exemplu, rețelele neuronale.

Tensorflow de asemenea deține o multitudine de modele pre-antrenate de învățare automată.

TensorFlow a fost dezvoltat de echipa Google Brain pentru utilizare internă a Google. A fost lansat sub licența Apache 2.0 pe 9 noiembrie 2015.

„TensorFlow este cel de-al doilea sistem Google Brain. Versiunea 1.0.0 a fost lansată pe 11 februarie 2017. În timp ce implementarea de referință se rulează pe dispozitive unice, TensorFlow poate rula pe mai multe CPU și GPU-uri (cu extensii opționale CUDA și SYCL pentru calcul de uz general pe unitățile de procesare grafică). TensorFlow este disponibil pe Linux pe 64 de biți, macOS, Windows și platforme de calcul mobil, inclusiv Android și iOS.

Arhitectura sa flexibilă permite desfășurarea ușoară a calculului pe o varietate de platforme (CPU, GPU, TPU) și de la computere desktop la grupuri de servere la dispozitive mobile și device-uri mici.

Calculul TensorFlow sunt exprimate sub formă de grafice de flux de date. Denumirea TensorFlow provine din operațiunile pe care astfel de rețele neuronale le efectuează pe tablouri de date multidimensionale, care sunt denumite tensori. În cadrul Conferinței Google I / O din iunie 2016, Jeff Dean a declarat că 1.500 de repositorye pe GitHub menționau TensorFlow, dintre care doar 5 erau de la Google.

În ianuarie 2018, Google au anunțat TensorFlow 2.0. În martie 2018, Google a anunțat TensorFlow.js versiunea 1.0 pentru învățare automată în JavaScript și TensorFlow Graphics pentru învățare profundă în grafică pe computer.

În mai 2016, Google și-a anunțat unitatea de procesare a tensiunii (TPU), un circuit integrat specific unei aplicații (un cip hardware) construit special pentru învățarea mașinilor și adaptat pentru TensorFlow. TPU este un accelerator AI programabil conceput pentru a oferi un randament ridicat de aritmetică de precizie redusă (de exemplu, pe 8 biți) și orientat către utilizarea sau rularea modelelor, mai degrabă decât să le antreneze. Google a anunțat că administrează TPU în centrele lor de date de mai bine de un an și le-a găsit să ofere o comandă de mărime performanțe optimizate mai bine per watt pentru învățarea mașinii.

În mai 2017, Google a anunțat a doua generație, precum și disponibilitatea TPU-urilor în Google Compute Engine. TPU-urile din a doua generație oferă până la 180 de teraflopi de performanță, iar atunci când sunt organizate în grupuri de 64 de TPU, oferă până la 11,5 petaflops.

În mai 2018, Google a anunțat cea de-a treia generație de TPU care livrează până la 420 de teraflop de performanță și 128 GB HBM. Cloud TPU v3 Pods oferă peste 100 petaflop-uri de performanță și 32 TB HBM.

În februarie 2018, Google a anunțat că pun la dispoziție TPU-uri în versiune beta pe platforma Google Cloud.”^[5]

2.7.4 Sentence Transformers

Sentence Transformers reprezintă un proiect pentru folosirea unor modele pre-antrenate pe diferite seturi de date sub diferite metode. Aceste modele sunt antrenate fin folosind diferite rețele neuronale pentru ca apoi să producă rezultate care să fie folosite în diferite scenarii nesupravegheate cum ar fi: similaritatea textuală folosind cosine-similarity, semantic search, etc. Acest proiect este folosit pentru a obține un model pre-antrenat fin pentru a putea obține rezultate mai bune.

2.7.5 Kaagle

Kaagle reprezintă o comunitate pentru practicanții de învățare automată. În cadrul acestui proiect, aceasta a fost folosită pentru inspirație, cât și pentru notebook-urile sale online care permit rularea codului și deci, antrenarea datelor.

2.8 Transformer

„*Transformer* este un model profund de învățare a mașinilor, introdus în 2017, utilizat în principal în domeniul procesării limbajului natural. Acesta a fost propus pentru prima dată în documentul “Attention is All You Need”. Este o arhitectură ce și propune să rezolve sarcini de la o secvență la alta, în timp ce gestionează cu ușurință dependențele pe distanțe lungi. Practic, ideea din spatele acestuia este să devină primul model de transducție care se bazează în întregime pe atenția de sine pentru a calcula reprezentările de intrare și ieșire fără a utiliza rețele neuronale recurente sau convoluțiuni, unde prin transducție înțelegem conversia dintre datele de intrare în date de ieșire.”

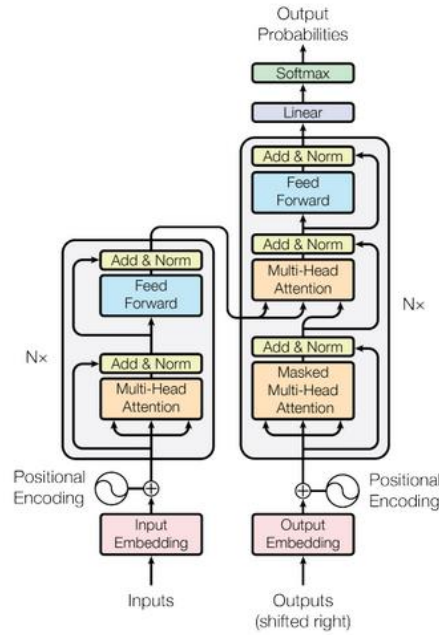


Figura 2 Arhitectura unui transformer

Encoderul conține un layer numit Multi-Head Attention urmat de alt layer numit Feed Forward Neural Network. Decoderul, pe de altă parte, conține layerele menționate anterior cât și un extra layer numit Masked Multi-Head Attention. **Codurile de codare și de decodificare sunt de fapt mai multe codificatoare și decodificatoare identice stivuite una peste alta.** Atât stiva codificator, cât și stiva decodificator au același număr de unități.

Blocurile de bază ale transformatorului sunt unități de atenție la nivel de produs. Când o propoziție este trecută într-un model Transformer, ponderile de atenție sunt calculate între fiecare simbol simultan. Stratul de atenție produce încorporări pentru fiecare simbol în context care conțin informații nu numai despre simbolul în sine, ci și o combinație ponderată de alte jetoane relevante ponderate de greutatea de atenție.

Calculul atenției pentru toate token-urile poate fi exprimat ca un singur calcul matricial mare, care este util pentru antrenament datorită optimizărilor operaționale matriciale care fac calculele rapide ale matricelor. Matricele Q , K și V sunt definite ca matrice în care rândurile sunt vectorii k_i , q_i respectiv. V_i .

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

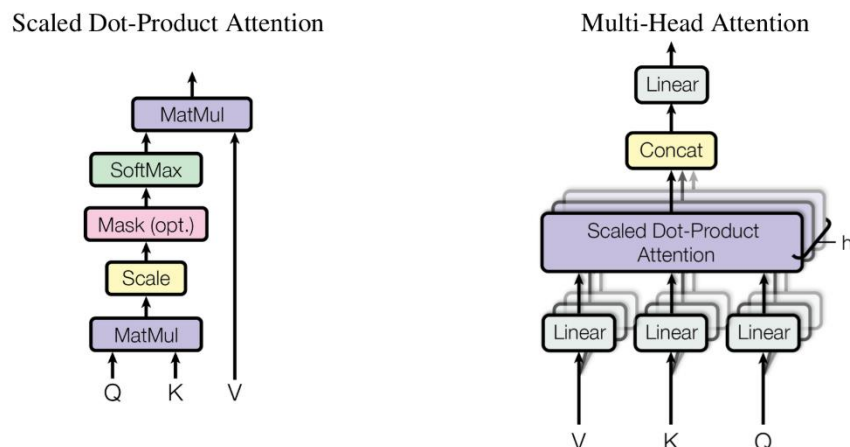


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Figura 3 Scaled Attention vs Mutli Head Attention

Aceste matrice Q, K și V sunt diferite pentru fiecare poziție a modulelor de atenție din structură, în funcție de faptul că sunt în codificator, decodificator sau în interiorul codificatorului și decodificatorului. Motivul este că dorim să participăm fie la întreaga secvență de intrare a codificatorului, fie la o parte a secvenței de intrare a decodificatorului. Modulul de atenție cu mai multe capete care conectează codificatorul și decodificatorul se va asigura că se ia în considerare secvența de intrare a codificatorului împreună cu secvența de intrare a decoderului până la o anumită poziție.

După capetele cu mai multe atenții atât în codificator, cât și în decodificator, avem un strat de avans în sens în sens punctual. Această mică rețea de avans are parametri identici pentru fiecare poziție, care poate fi descrisă ca o transformare liniară separată, identică a fiecărui element din secvența dată.

Cum se antrenează?

Să presupunem că vrem să antrenăm un model pentru o traducere. Știm că pentru a instrui un model pentru sarcinile de traducere avem nevoie de două propoziții în limbi diferite, care sunt traduceri între ele. După ce vom avea multe perechi de propoziții, putem începe să ne antrenăm modelul. Să zicem că vrem să traducem franceza în germană. Intrarea noastră codificată va fi o propoziție franceză, iar intrarea pentru decodificator va fi o propoziție germană. Cu toate acestea, intrarea decodificatorului va fi deplasată spre dreapta cu o singură poziție, deoarece nu dorim ca modelul nostru să învețe cum să ne copieze intrarea decodificatorului în timpul antrenamentului, dar vrem să aflăm că, dată fiind secvența

codificatorului și o anumită secvență de decodificator, care a fost deja văzută de model, prezicem următorul cuvânt / caracter.

Dacă nu schimbăm secvența decodificatorului, modelul învață să „copieze” pur și simplu intrarea decodificatorului, întrucât cuvântul / caracterul țintă pentru poziția i ar fi cuvântul / caracterul i din intrarea decodificatorului. Astfel, prin schimbarea intrării decodificatorului într-o singură poziție, modelul nostru trebuie să prezică cuvântul / caracterul țintă pentru poziția i doar că a văzut cuvântul / caracterele $1, \dots, i-1$ în secvența decodificatorului. Acest lucru împiedică modelul nostru să învețe sarcina de copiere / lipire. Completăm prima poziție a intrării decodificatorului cu un token de început de propoziție, deoarece locul respectiv ar fi gol, din cauza deplasării la dreapta. În mod similar, atașăm un simbol de sfârșit de propoziție la secvența de intrare a decodificatorului pentru a marca sfârșitul acelei secvențe și este de asemenea anexat la propoziția de ieșire țintă. Într-o clipă, vom vedea cum este util acest lucru pentru a deduce rezultatele.

În plus față de deplasarea spre dreapta, Transformatorul aplică o mască la intrarea din primul modul de atenție cu mai multe capete, pentru a evita să se vadă potențiale elemente de secvență „viitoare”.^[6]

Pe scurt, metoda step-by-step ar fi

- Introducem secvența completă de codificare (propoziție franceză) și ca intrare a decodificatorului, luăm o secvență goală cu doar un token de început de propoziție pe prima poziție. Aceasta va produce o secvență în care vom lua doar primul element.
- Acest element va fi completat în a doua poziție a secvenței noastre de introducere a decodificatorului, care are acum un token de început de propoziție și un prim cuvânt / caracter în el.
- Introducem în model atât secvența de codare cât și noua secvență de decodificare. Preluăm al doilea element al ieșirii și îl plasăm în secvența de intrare a decodificatorului.
- Repetăm acest lucru până când prezicem un simbol de sfârșit de propoziție, care marchează sfârșitul traducerii.

Transformer-ul are o aplicabilitate largă, însă este cel mai întâlnit în NLP, sub diferite modele pre-antrenate. Cele folosite de mine sunt BERT și RoBERTa.

2.8.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) este o tehnica destinată PLN pre-antrenată și dezvoltată de către compania Google. BERT a fost creat cu scopul de a înțelege căutările utilizatorilor de pe motorul de căutare al celor de la Google. Acesta a folosit 2 dataseturi în pre-antrenare

și anume *BookCorpus* și *Wikipedia*. Când acesta a fost lansat, a atins un nivel foarte bun al performanței pe câteva task-uri de înțelegere al limbajului natural, și anume

- GLUE (tradusă ca General Language Understanding Evaluation) pe cele 9 task-uri.
- SquAD (rezultate foarte bune pe dataset-ul Stanford Question Answering Dataset).
- SWAG (Situations with Adversarial Generations)

Spre deosebire de modelele anterior lansate, BERT este o reprezentare profundă bidirecțională, nesupravegheată, pre-instruită folosind doar un corpus simplu. Modelele fără context, cum ar fi word2vec sau GloVe generează o reprezentare de încorporare a unui singur cuvânt pentru fiecare cuvânt din vocabular, unde OAR ține cont de contextul pentru fiecare apariție a unui cuvânt dat. De exemplu, în timp ce vectorul pentru „running” va avea aceeași reprezentare vectorială word2vec pentru ambele apariții ale sale în propozițiile „He is running a company.” și „He is running in a marathon.”, BERT va oferi o încorporare contextualizată care va fi diferit în funcție de propoziție.

De aceea, am considerat ca BERT este un candidat bun pentru testarea în task-ul meu de detectare a similarității în text.

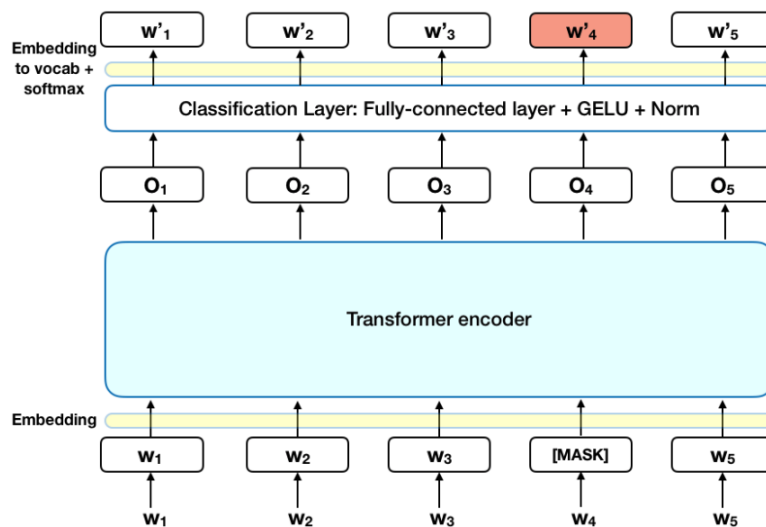


Figura 4 Structura BERT

Cum funcționează BERT?

BERT folosește Transformer, un mecanism de atenție care învață relațiile contextuale între cuvinte (sau sub-cuvinte) dintr-un text. În forma sa de vanilie, Transformer include două mecanisme separate - un codificator care citește introducerea textului și un decodificator care produce o predicție pentru sarcină.

Deoarece obiectivul BERT este acela de a genera un model de limbă, este necesar doar mecanismul codificatorului. Lucrările detaliate ale Transformer sunt descrise într-o lucrare de Google.

Spre deosebire de modelele direcționale, care citesc introducerea textului secvențial (de la stânga la dreapta sau de la dreapta la stânga), codificatorul Transformer citește întreaga secvență de cuvinte simultan. Prin urmare, este considerat bidirecțional, deși ar fi mai exact să spunem că nu este direcțional. Această caracteristică permite modelului să învețe contextul unui cuvânt bazat pe toate împrejurimile sale (stânga și dreapta cuvântului).

Figura de mai sus este o descriere la nivel înalt a codificatorului Transformator. Intrarea este o secvență de jetoane, care sunt mai întâi încorporate în vectori și apoi procesate în rețeaua neuronală. Ieșirea este o secvență de vectori de dimensiunea H , în care fiecare vector corespunde unui jeton de intrare cu același index.

Când se formează modele de limbaj, există o provocare de a defini un obiectiv de predicție. Multe modele prezic următorul cuvânt într-o secvență (de exemplu, „Copilul a venit acasă de la ____”), o abordare direcțională care limitează în mod inerent învățarea în context. Pentru a depăși această provocare, BERT folosește două strategii de formare:

1. Masked LM (MLM)

Înainte de a introduce secvențe de cuvinte în BERT, 15% din cuvintele din fiecare secvență sunt înlocuite cu un simbol [MASK]. Modelul încearcă apoi să prezice valoarea inițială a cuvintelor mascate, pe baza contextului oferit de celelalte cuvinte care nu sunt mascate din secvență. În termeni tehnici, predicția cuvintelor de ieșire necesită:

- Adăugarea unui strat de clasificare pe partea de sus a ieșirii codificatorului.
- Înmulțirea vectorilor de ieșire cu matricea de încorporare, transformându-i în dimensiunea vocabularului.
- Calcularea probabilității fiecărui cuvânt din vocabular cu softmax.

Funcția de pierdere BERT ia în considerare doar predicția valorilor mascate și ignoră predicția cuvintelor care nu sunt mascate. În consecință, modelul converge mai lent decât modelele direcționale, caracteristică care este compensată de conștientizarea contextului crescut pe care acesta îl are.

2. Predicția propoziției următoare (NSP)

În procesul de instruire BERT, modelul primește perechi de propoziții ca input și învață să prezice dacă a doua propoziție din pereche este propoziția ulterioară din documentul original. În timpul antrenamentului, 50% din intrări sunt o pereche în care a doua propoziție este propoziția ulterioară din documentul original, în timp ce în cealaltă 50% o propoziție aleatorie din corpus este aleasă ca a doua propoziție. Presupunerea este că propoziția întâmplătoare va fi deconectată de la prima propoziție.

Pentru a ajuta modelul să facă distincția între cele două propoziții la antrenament, introducerea este procesată în felul următor înainte de a intra în model:

- Un simbol [CLS] este inserat la începutul primei propoziții și se introduce un simbol [SEP] la sfârșitul fiecărei propoziții.
- La fiecare simbol, se adaugă o propoziție care indică propoziția A sau propoziția B. Propozițiile incorporate sunt similare în conceptul încorporat în token-uri cu un vocabular de 2.
- La fiecare token se adaugă o încorporare care conferă poziția sa din secvență.

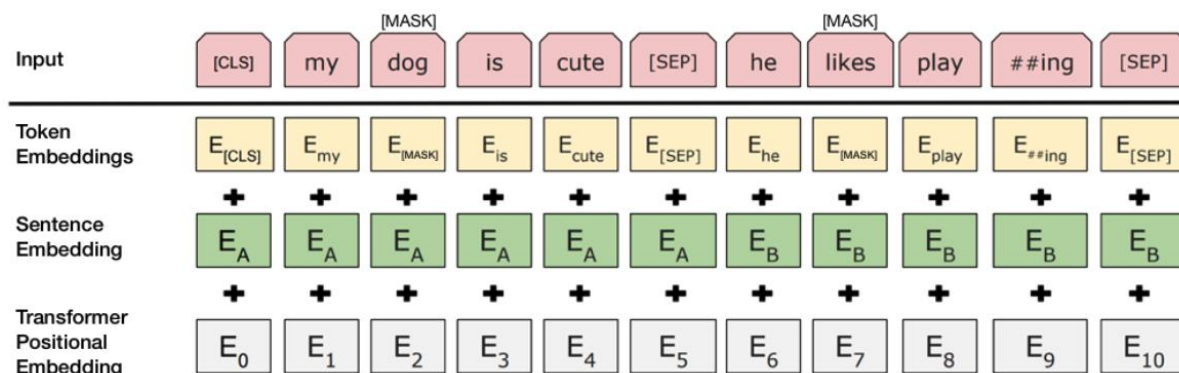


Figura 5 BERT Embeddings

Pentru a putea să prezică dacă a doua propoziție este legată de prima, se parcurg următorii pași:

- Întreaga propoziție trece prin modelul Transformer.
- Output-ul token-ului CLS este transformat în un vector de dimensiune 2x1, folosind un simplu strat de clasificare
- Se calculează probabilitatea variabilei IsNextSequence folosind softmax.

Când modelul BERT este antrenat, atât partea de Masked LM cât și partea de Predicție a propoziției următoare sunt antrenate împreună, cu scopul de a minimiza pierderea acestor metode.

Cum folosim BERT? (Fine-Tuning)

Utilizarea BERT pentru o sarcină specifică este relativ simplă:

BERT poate fi utilizat pentru o mare varietate de sarcini de limbaj, adăugând doar un strat mic la modelul de bază:

- Sarcinile de clasificare, cum ar fi analiza sentimentului, se realizează în mod similar clasificării NSP, adăugând un strat de clasificare în partea de sus a ieșirii Transformatorului pentru simbolul [CLS].
- În sarcini de răspuns la întrebări (de ex. SQuAD v1.1), software-ul primește o întrebare cu privire la o secvență text și este obligat să marcheze răspunsul în secvență. Folosind BERT, un model Q&A poate fi instruit învățând doi vectori suplimentari care marchează începutul și sfârșitul răspunsului.
- În recunoaștere a entităților (NER), software-ul primește o secvență text și este obligat să marcheze diferitele tipuri de entități (persoană, organizație, dată etc.) care apar în text. Folosind BERT, un model NER poate fi instruit alimentând vectorul de ieșire al fiecărui token într-un strat de clasificare care prezice eticheta NER.

Dimensiunea modelului contează, chiar și la scară uriașă. BERT_large, cu 345 milioane de parametri, este cel mai mare model de acest gen. Este demonstrativ superioară la sarcinile la scară mică față de BERT_base, care utilizează aceeași arhitectură cu „doar” 110 milioane de parametri.

Cu suficiente date de instruire, mai mulți pași de instruire înseamnă o precizie mai mare. De exemplu, în sarcina MNLI, precizia BERT_base se îmbunătățește cu 1,0% atunci când este antrenat pe pași de 1M (dimensiune lot de 128.000 de cuvinte), comparativ cu pași de 500K cu aceeași dimensiune a lotului.

Abordarea bidirecțională (MLM) a BERT converg mai lent decât abordările de la stânga la dreapta (deoarece doar 15% din cuvinte sunt prevăzute în fiecare lot), dar formarea bidirecțională întrece în

continuare formarea de la stânga la dreapta după un număr mic de etape de pre-instruire.”^[7]

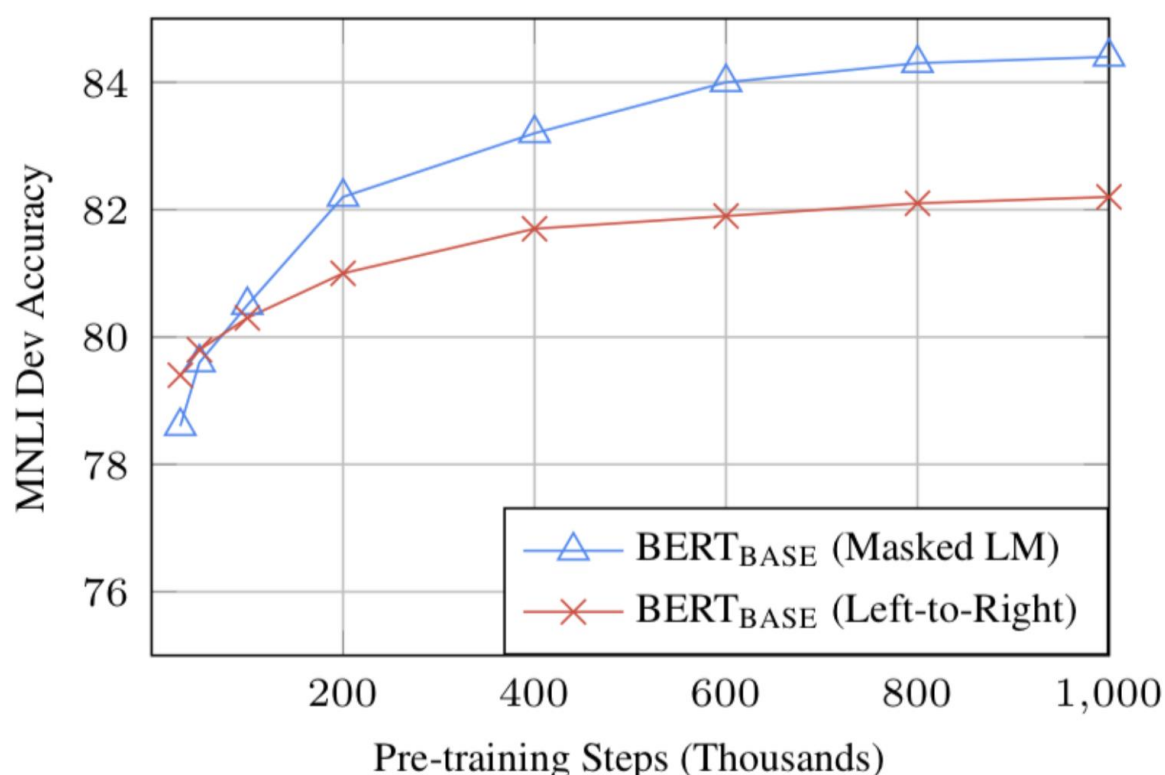


Figura 6 Masked LM vs Left To Right

BERT reprezintă, fără îndoială, un progres în utilizarea învățării automate pentru procesarea limbajului natural. Faptul că este abordabil și permite o ajustare rapidă va permite probabil o gamă largă de aplicații practice în viitor.

2.9 GloVe

„GloVe”, creat de la Global Vectors, este un algoritm pentru reprezentarea distribuită a cuvintelor. Modelul este un algoritm de învățare nesupravegheat pentru obținerea reprezentărilor vectoriale pentru cuvinte. Acest lucru este obținut prin cartografierea cuvintelor într-un spațiu semnificativ în care distanța dintre cuvinte este legată de similaritatea semantică. Antrenarea se realizează pe statistici globale agregate de coincidență cuvânt-cuvânt dintr-un corpus, iar reprezentările rezultate prezintă substructuri liniare interesante ale spațiului vectorului cuvânt. Este dezvoltat ca un proiect open-source la Stanford. Ca model de regresie log-bilineară pentru învățarea nesupervizată a reprezentărilor de cuvinte, combină caracteristicile a două familii de modele, și anume factorizarea matricei globale și metodele de fereastră a contextului local.

GloVe poate fi de asemenea folosit pentru a găsi relații între cuvinte precum sinonime și coduri poștale. Este de asemenea folosit de către modelul *SpaCy* pentru a defini relații semantice pentru word

embeddings/ feature vectors ca ulterior să fie folosite metode precum similaritatea cosinus sau chiar distanța euclidiană pe aceste relații.”^[8]

GloVe încearcă să atingă 2 scopuri

- Să creeze word embeddings care să dețină sens în spațiul vectorial
- Profită de statisticile globale de numărare, în loc de informațiile locale

Spre deosebire de word2vec - care învață prin transmiterea de propoziții - GloVe învață pe baza unei matrice de coincidență și antrenează vectori de cuvinte, astfel încât diferențele lor prezic raporturi de coincidență.

Cum folosim GloVe?

Primul pas este construirea unei matrice de coincidență. De asemenea, GloVe ține cont de contextul local, calculând matricea de coincidență folosind o dimensiune fixă a ferestrei (cuvintele sunt considerate că apar simultan atunci când apar împreună într-o fereastră fixă). De exemplu, propoziția "The cat sat on the mat" cu o dimensiune 2 ar fi convertită în matricea de coincidență:

Tabel 1 Matricea de coincidență

	the	cat	sat	on	mat
the	2	1	2	1	1
cat	1	1	1	1	0
sat	2	1	1	1	0
on	1	1	1	1	1
mat	1	0	0	1	1

Observați cum matricea este simetrică: aceasta se întâmplă pentru că atunci când cuvântul „cat” apare în contextul „sat”, se întâmplă opusul (cuvântul „sat” care apare în contextul „cat”).

Acum, întrebarea este cum să conectăm vectorii cu statisticile calculate mai sus. Principiul care stă la baza lui GloVe poate fi declarat după cum urmează: raporturile de coincidență între două cuvinte într-un context sunt puternic legate de sens.

Acest lucru sună dificil, dar ideea este foarte simplă. Luați cuvintele „gheață” și „abur”, de exemplu. Gheața și aburul diferă de starea lor, dar sunt aceleași prin faptul că sunt ambele forme de apă. Prin urmare, ne-am aștepta ca cuvintele legate de apă (cum ar fi „apă” și „umed”) să apară în mod egal în contextul „gheață” și „abur”. În schimb, cuvinte precum „rece” și „solid” ar apărea probabil lângă „gheață”, dar nu ar apărea lângă „abur”.

Algoritmul GloVe pe scurt, ar fii

- Dat un vocabular, un corpus, o dimensiune a ferestrei (N) și o coincidență minimă, algoritmul ia o fereastră glisantă de dimensiunea N și o trece prin întregul corpus, numărând coincidențele fiecărui cuvânt cu fiecare alt cuvânt. Rezultatul este matricea rară X_{ij} .
- Inițializarea parametrilor modelului. Acestea sunt matricea vectorului W de mărime $(V \times D)$, unde V denumește mărimea vocabularului și D semnifică dimensiunile vectorului specificate; și un vector de părtinire pentru fiecare termen. Fiecare termenul este inițiat la întâmplare în intervalul $(-0,5, 0,5)$.
- Co-aparițiile sunt amestecate și algoritmul calculează funcția de cost asociată cu faza inițială a algoritmului.
- Pentru fiecare iterație, sunt obținuți în conformitate cu parametrii gradienti ai funcției de cost J , iar parametrii sunt actualizați corespunzător la o rată de învățare.

2.10 TF-IDF

„În recuperarea informațiilor, **tf-idf** sau **TFIDF**, abreviere din engleză *term frequency-inverse document frequency*, însemnând *frecvența termenului-frecvența inversă în documente*, este o măsură statistică numerică destinată cuantizării importanței unui cuvânt într-un document într-o colecție sau corpus. Această măsură este adesea folosită ca factor de ponderare în căutările din recuperarea informațiilor, text mining și modelarea utilizatorului. Valoarea tf-idf crește proporțional cu numărul de apariții ale unui cuvânt într-un document și este contrabalansat de numărul de documente din corpus care conțin cuvântul, ceea ce ajută la ponderarea necesară din cauză că unele cuvinte apar mai frecvent în general. Tf-idf este una dintre cele mai populare metode de ponderare a termenilor astăzi, iar 83% din sistemele de recomandare bazate pe text din bibliotecile digitale folosesc tf-idf.

Tf-idf este produsul a două mărimi statistice, *frecvența termenilor* și *frecvența inversă în documente*. Există diverse metode pentru determinarea valorilor exacte ale acestor mărimi statistice. De asemenea, este o formulă care își propune să definească importanța unui cuvânt sau unei fraze cheie într-un document sau o pagină web.

În cazul **frecvenței termenului** $tf(t,d)$, cea mai simplă opțiune este utilizarea *numărării brute* a unui termen într-un document. Cu alte cuvinte, numărul de apariții ale termenului t în documentul d . Dacă $f_{t,d}$ este numărul brut de apariții, atunci cea mai simplă schemă tf este $tf(t,d) = f_{t,d}$.

Alte posibilități includ:

- „Frecvențe” booleene: $tf(t,d) = 1$ dacă t apare în d și 0 în caz contrar;
- Frecvența termenului ajustată pentru lungimea documentului: $f_{t,d} \div$ (numărul de cuvinte în d);
- Frecvența scalată logaritmic: $tf(t,d) = \log(1 + f_{t,d})$;
- Frecvența augmentată, pentru a preveni erori în cazul documentelor mai lungi, de exemplu, frecvența brută împărțită frecvența brută a termenului care apare cel mai des în document:

$$tf(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$tf(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

Tabel 1: Variante de pondere a frecvenței termenilor(tf)

Schema de ponderare	Valoare tf
binar	0,1
Numărare brută	$f_{t,d}$
frecvența termenilor	$\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$
normalizare logaritmică	$\log(1 + f_{t,d})$
normalizare dublă 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
normalizare dublă K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Frecvența inversă în documente este o măsură a cât de multă informație este conținută de termen, adică dacă e comun sau rar în colecția de documente. Este valoarea logaritmată a inversului fracției dintre numărul de documente care conțin cuvântul și numărul total de documente:

$$\text{idf}(t,D)=\log \frac{N}{|\{d \in D: t \in d\}|}$$

unde

- N : numărul total de documente în corpus $N=|D|$
- $|\{d \in D: t \in d\}|$: numărul de documente în care apare termenul t (de exemplu, $\text{tf}(t,d) \neq 0$). Dacă termenul nu este în corpus, acest lucru va duce la o împărțire la zero. Prin urmare, de obicei se modifică numitorul astfel încât să fie $1+|\{d \in D: t \in d\}|$.

Așadar, tf-idf este calculat ca:

$$\text{tfidf}(t, d, D)=\text{tf}(t,d) \cdot \text{idf}(t,D)$$

O pondere ridicată a tf-idf este atinsă de o frecvență mare a termenului (în documentul dat) și o frecvență scăzută în documente în cadrul întregii colecții de documente. Ponderile tind astfel să filtreze termenii obișnuiți. Deoarece raportul din funcția logaritmică a idf este întotdeauna mai mare sau egal cu 1, valoarea idf (și tf-idf) este mai mare sau egală cu 0. Pe măsură ce un termen apare în mai multe documente, raportul din logaritm se apropie de 1, aducând idf și tf-idf mai aproape de 0.”^[9]

Tabel 2: Variante de pondere a idf in documente

Schema de ponderare	Valoare idf ($n_t= \{d \in D: t \in d\} $)
unar	1
frecvența inversă în documente	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
frecvența inversă în documente netedă	$\log \left(\frac{N}{1+n_t} \right) + 1$
frecvența inversă în documente maximă	$\log \left(\frac{\max_{\{t \in d\}} n_{t'}}{1+n_t} \right)$
frecvența inversă în documente probabilistică	$\log \frac{N - n_t}{n_t}$

Tabel 3: Scheme de ponderare recomandate pentru tf-idf

Schema de ponderare	Pondere a termenului în documente	Ponderea termenului interogant
1	$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \cdot \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
2	$1 + \log f_{t,d}$	$\log \left(1 + \frac{N}{n_t}\right)$
3	$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

2.11 Similaritatea cosinus

“O abordare frecvent utilizată pentru a găsi documente similare se bazează pe numărarea numărului maxim de cuvinte comune între documente. Această abordare are un defect inerent. Adică, pe măsură ce dimensiunea documentului crește, numărul de cuvinte comune tind să crească chiar dacă documentele vorbesc despre subiecte diferite.

Similaritatea cosinus ajută la depășirea acestui defect fundamental în abordarea „numărați-comun-cuvinte” sau la distanța euclidiană. Similaritatea cosinus este o metrică utilizată pentru a determina cât de similare sunt documentele indiferent de mărimea lor. Matematic, măsoară cosinusul unghiului dintre doi vectori proiectați într-un spațiu multidimensional. În acest context, cei doi vectori despre care vorbesc sunt tablouri care conțin numărarea cuvintelor a două documente.

Ca metrică a similarității, cum diferă asemănarea cosinusului de numărul de cuvinte comune?

Când este desenat pe un spațiu multidimensional, unde fiecare dimensiune corespunde unui cuvânt din document, asemănarea cosinusului surprinde orientarea (unghiul) documentelor și nu mărimea.

Similaritatea cosinus este avantajoasă deoarece, chiar dacă cele două documente similare sunt departe de distanța euclidiană din cauza mărimii (cum ar fi, cuvântul „greier” a apărut de 50 de ori într-un document și de 10 ori în altul) acestea ar putea avea încă un unghi mai mic între ele. Cu cât unghiul este mai mic, mai mare este similaritatea.

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ is the dot product of the two vectors.

Figura 7 Formula Similarității Cosinus

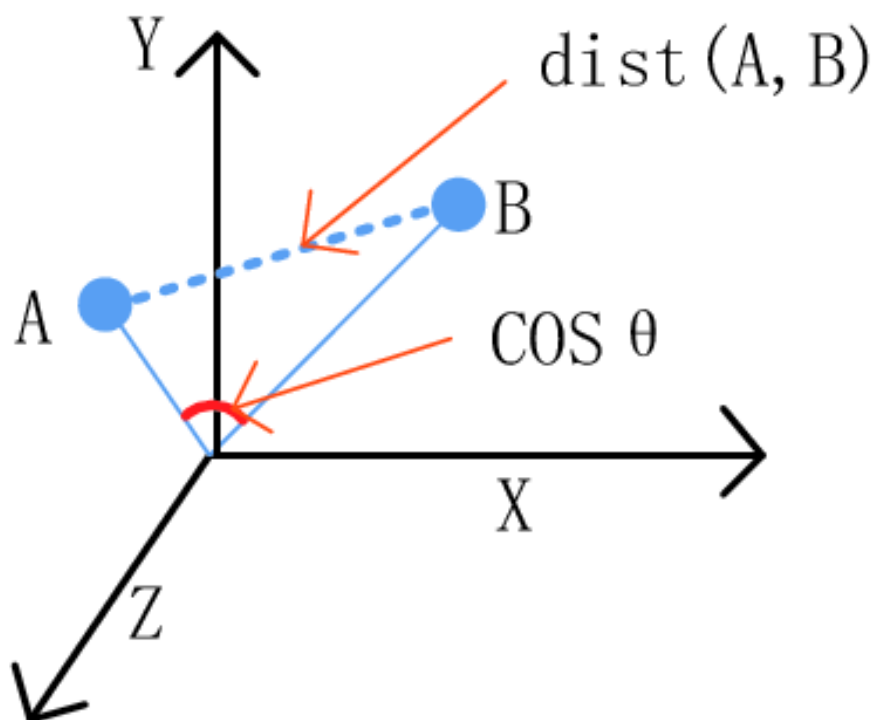


Figura 8 Similaritatea Cosinus

Cum funcționează Similaritatea Cosinus?

Să presupunem că aveți 3 documente bazate pe câțiva jucători de cricket – Sachin, Tendulkar și Dhoni. Două dintre documentele (A) și (B) sunt din paginile wikipedia de pe jucătorii respectivi, iar al treilea document (C) este un fragment mai mic din pagina de wikipedia a lui Dhoni.

The Three Documents and Similarity Metrics

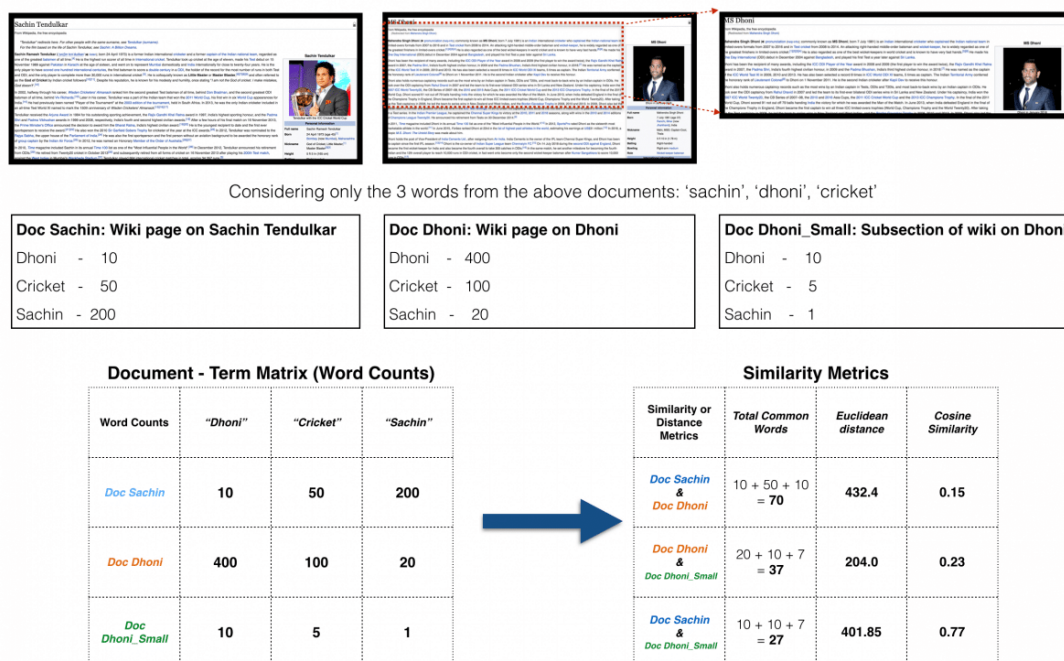


Figura 9 Cele 3 documente exemplu

După cum vedeți, toate cele trei documente sunt conectate printr-o temă comună - jocul Cricket.

Obiectivul nostru este să estimăm cantitativ asemănarea dintre documente. Pentru o mai ușoară înțelegere, să luăm în considerare doar primele 3 cuvinte comune dintre documente: „Dhoni”, „Sachin” și „Cricket”. Ne-am aștepta ca Doc A și Doc C, adică cele două documente de pe Dhoni să aibă o similaritate mai mare față de Doc A și Doc B, deoarece, Doc C este, în esență este un fragment din Doc A însuși. Cu toate acestea, dacă mergem după numărul de cuvinte comune, cele două documente mai mari vor avea cele mai comune cuvinte și, prin urmare, vor fi apreciate ca fiind cele mai similare, ceea ce este exact ceea ce dorim să evităm. Rezultatele ar fi mai congruente atunci când vom folosi scorul de asemănare cosinus pentru a evalua similitudinea.

Proiectăm documentele în un spațiu tri-dimensional, unde fiecare dimensiune este un număr de frecvență fie: „Sachin”, „Dhoni” sau „Cricket”. Când sunt reprezentate în acest spațiu, cele 3 documente sunt reprezentate astfel ca în figura 8:

Projection of Documents in 3D Space

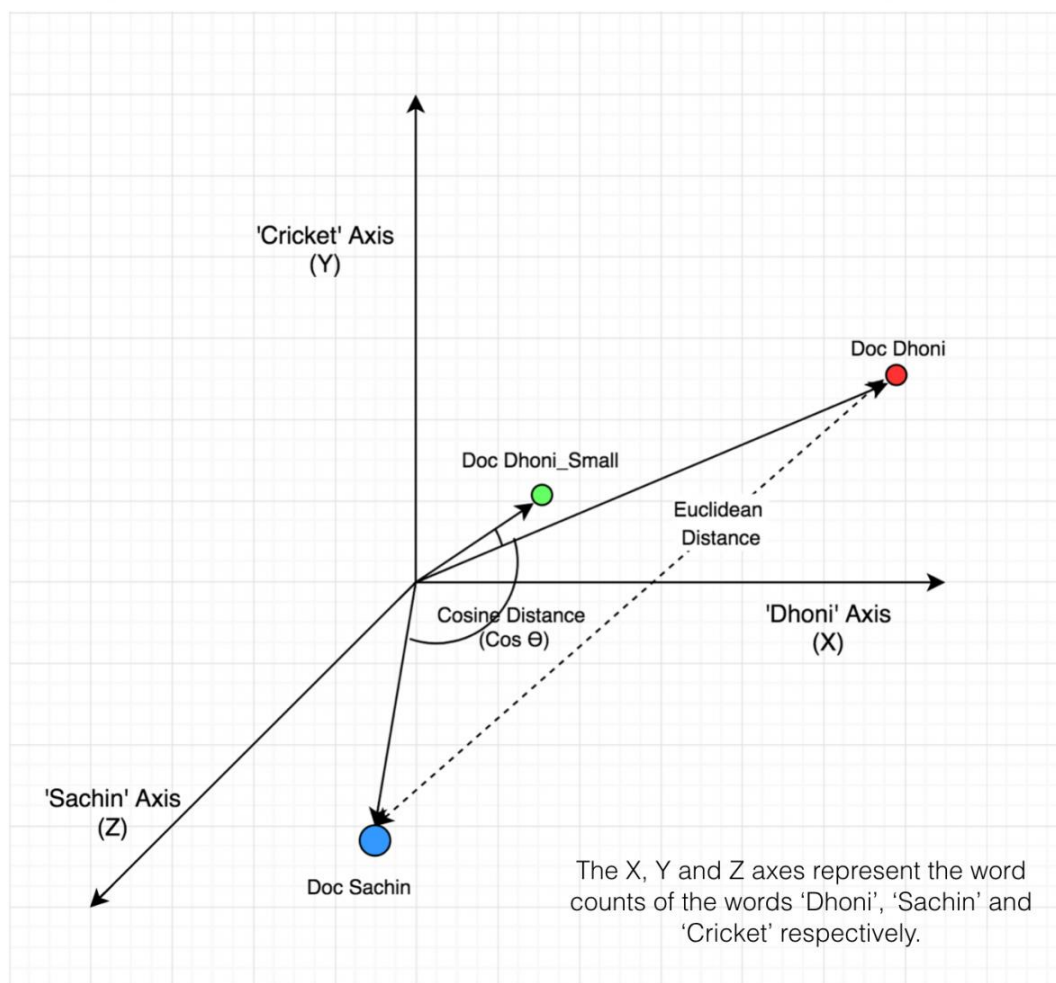


Figura 10 Proiectarea Documentelor

După cum puteți vedea, documentul Dhoni_Small și principalul document Dhoni sunt orientate mai aproape împreună în spațiul 3-D, chiar dacă sunt departe de magnitudine. Se dovedește, cu cât documentele sunt mai aproape de unghi, cu atât este mai mare similaritatea cosinus.”^[10]

Concluzionând, similaritatea cosinus este o metodă excelentă dacă dorim să utilizăm o metrică de similaritate care poate ajuta la clusterizarea sau clasificarea documentelor.

2.12 RoBERTa

RoBERTa se bazează pe strategia de mascare a limbii BERT, în care sistemul învață să prezice secțiuni de text ascunse în mod intenționat în exemple de limbă altfel neadnotate. RoBERTa, care a fost implementat în PyTorch, modifică hiperparametrele cheie în OAR, inclusiv eliminarea obiectivului de pre-antrenare a următoarei propoziții a lui BERT și formarea cu mini-loturi și rate de învățare mult mai

mari. Acest lucru permite modelului să îmbunătățească obiectivul de modelare a limbajului mascat în comparație cu BERT și duce la o mai bună performanță a sarcinilor în aval. De asemenea, explorăm instruirea RoBERTa pe un ordin de mărime mai multe date decât OAR, pentru o perioadă mai lungă de timp. Am utilizat seturi de date NLP existente neadnotate, precum și CC-News, un set nou extras din articole de știri publice.

După implementarea acestor modificări de proiectare, modelul a furnizat performanțe de ultimă generație la sarcinile MNLI, QNLI, RTE, STS-B și RACE și o îmbunătățire a performanței considerabilă pe etalonul GLUE. Cu un scor de 88,5, RoBERTa a ajuns pe poziția superioară pe clasamentul GLUE, egalând performanțele liderului anterior, XLNet-Large. Aceste rezultate subliniază importanța alegerilor de proiectare neexplorate anterior în formarea OART și ajută la dezangajarea contribuțiilor relative ale mărimii datelor, a timpului de pregătire și a obiectivelor de preliminară.”^[11]

3 DESIGN SI IMPLEMENTARE

3.1 Descrierea dataset-ului

În aceasta lucrare, am folosit 1 dataset pentru antrenarea modelelor BERT respectiv RoBERTa, și 1 data set pentru modelul Tf-idf & GloVe.

Primul dataset pe care BERT&RoBERTa au fost antrenate a fost corpusul SNLI, dataset ce conține 570.000 perechi de propoziții în limba engleză, scrise de om, etichetate manual pentru o clasificare echilibrată cu etichetele legătura, contradicție, neutru.

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

Figura 11 Sample din dataset

Al doilea dataset este reprezentat de către corpusul MultiNLI.

Corpusul Multi-Genre Natural Language Inference (MultiNLI) este o colecție de 433 k de perechi de propoziții adnotate cu informații legate de text. Corpusul este modelat pe corpusul SNLI, dar diferă prin faptul că acoperă o serie de genuri de texte vorbite și scrise și susține o evaluare distinctivă a generalizării între genuri. Corpusul a servit drept bază pentru sarcina comună a Atelierului RepEval 2017 la EMNLP din Copenhaga.

Aceste modele au fost furnizate de către o librărie externă, procesul de antrenare propriu zis durând extraordinar de mult.

În final, modele au fost tunate fin folosind un ultim dataset, și anume STS benchmark dataset. STS Benchmark cuprinde o selecție a seturilor de date engleze utilizate în sarcinile STS organizate în contextul SemEval între 2012 și 2017. Selecția de seturi de date include text din titluri de imagine, titluri de știri și forumuri pentru utilizatori.

Aceste modele au fost antrenate pe 3 seturi, deoarece RoBERTa cat și BERT produc Word-Embeddings relativ slabe daca acestea nu sunt antrenate.

```
2020-06-19 15:54:05 - Read in embeddings file glove.6B.380d.txt.gz
Load Word Embeddings: 400000Embeddings [00:38, 10526.22Embeddings/s]
2020-06-19 15:54:50 - 17027 of 400001 words without a weighting value. Set weight to 15.48518864185265
2020-06-19 15:54:50 - Use pytorch device: cpu
2020-06-19 15:54:50 - Read STSBenchmark train dataset
Convert dataset: 100% | 5749/5749 [00:00:00:00, 34948.68it/s]
2020-06-19 15:54:50 - Num sentences: 5749
2020-06-19 15:54:50 - Sentences 0 longer than max_sequence_length: 0
2020-06-19 15:54:50 - Sentences 1 longer than max_sequence_length: 0
2020-06-19 15:54:50 - Read STSBenchmark dev dataset
Convert dataset: 100% | 1500/1500 [00:00:00:00, 64148.13it/s]
2020-06-19 15:54:50 - Num sentences: 1500
2020-06-19 15:54:50 - Sentences 0 longer than max_sequence_length: 0
2020-06-19 15:54:50 - Sentences 1 longer than max_sequence_length: 0
2020-06-19 15:54:50 - Warmup-steps: 180
Iteration: 100% | 180/180 [00:02:00:00, 83.84it/s]
2020-06-19 15:54:52 - Evaluation the model on dataset after epoch 0:
Convert Evaluating: 100% | 47/47 [00:00:00:00, 304.80it/s]
```

Figura 12 Detalii dataset STSBenchmark

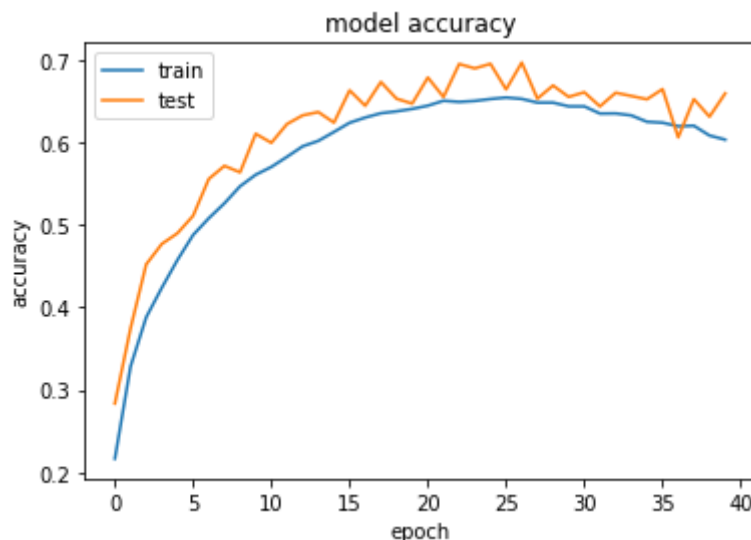
După cum putem observa din Figura 10, STSBenchmark conține un număr de 5749 de propoziții pentru antrenare și 1500 de propoziții pentru dev. Aceasta este un dataset bineînțeles împărțit în 3 module, după clasicele procentaje de 70%, 15%, 15% pentru train, dev si test.

Modelul GloVe, a fost pre dispus de asemenea de o librărie externa, si antrenat folosind un dataset cu frecventa cuvintelor de pe site ul Wikipedia.

Ce este un epoch?

„Un Epoch se întâmplă atunci când un întreg set de date este transmis înainte și înapoi prin rețeaua neuronală o singură dată. Din moment ce o epocă este prea mare pentru a se alimenta computerului, trebuie împărțit în câteva mai mici loturi. Pe măsură ce numărul de epoci crește, se modifică mai multe ori greutatea în rețea neuronală și curba merge de la adecvarea la cea optimă la cea superioară.”^[13]

Modelele au fost antrenate pe un număr diferit de epochs, salvate temporar de fiecare dată după fiecare epoch, în cazul în care pierderea validării scădea comparată cu cea anterioară, cu un batch size de 16/32 de valori.



3.2 Antrenare și rezultate

3.2.1 WordEmbeddings folosind GloVe + tf/idf.

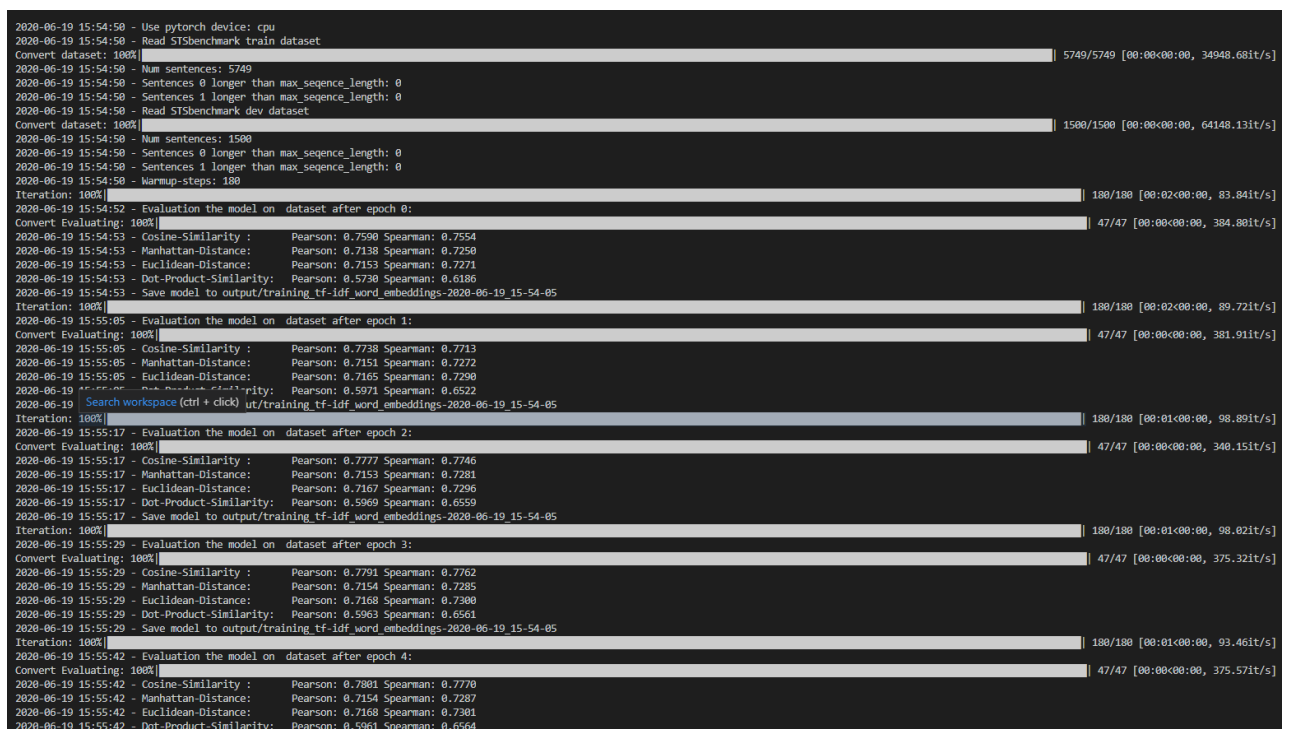


Figura 14 Model Antrenare GloVe-TfIdf

Am folosit un batch size de valoare 32, citind bineînțeles dataset-ul Wikipedia Document Frequencies.

Modelul a fost salvat în folder-ul de output de îndată ce acesta a fost antrenat.

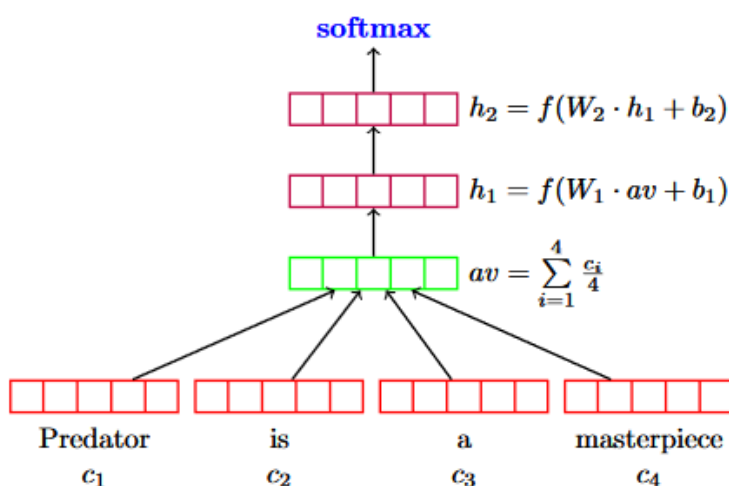
Am folosit ca algoritm de producere a word embeddings GloVe, am descărcat GloVe și în conformitate cu acesta am produs maparea token-urilor la tradiționalul word embedding.

După care, am calculat greutatea word embeddings-urilor folosind valorile IDF (Inverse Document Frequency) generate, citind de pe toate liniile valorile frecvențelor. Pentru aceste valori, am folosit un document Wikipedia în care acestea erau numerotate, pentru simplitate. Pentru fiecare cuvânt din vocabularul tokenizer-ului, trebuie specificată o valoare a greutății, care apoi este înmulțită cu aceea valoare.

Am inițializat modelul folosind clasa WordWeights, după care am aplicat mean pooling pentru a obține un sentence vector de mărime fixă. Deoarece am folosit mean-pooling, am fost nevoit să implementez 2 rețele neuronale DAN de tipul feed-forward, mean-pooling și aceste rețele produc împreună astfel o clasificare mult mai bună a respectivelor propoziții, pentru a putea fi puse în vectorii respectivi. Acestea sunt antrenamente în plus, pentru o prezicere a rezultatului final mai bună.

Un model profund neordonat, care obține o acuratețe de ultimă generație la sarcinile și sarcinile la nivel de document, cu foarte puțin timp de pregătire funcționează în trei etape:

- preluată media vectorială a încorporărilor asociate cu o secvență de intrare de tokens.
- pasată această medie prin un nivel feed-forward.
- execută clasificarea pe reprezentarea ultimelor layere.



DAN antrenează modelul în un timp scurt, dar cu o acuratețe mai scăzută decât altă alternativă.

De asemenea, pentru a putea antrena modelul, avem nevoie de aceste rețele neuronale.

După care, folosind clasa `SentenceTransformer`, putem salva modelul folosind aceste artificii făcute în plus. Singurul lucru care mai trebuie făcut, este dat datasetul pentru antrenare și validare.

În primul rând, datasetul nostru trebuie să fie convertit în un obiect de tipul `DataLoader`, pentru a putea fi evaluat ulterior. Datasetul este de asemenea împărțit în `train` și `dev`.

Pentru partea de citire a dataset-ului de `train` avem nevoie de 3 clase:

- ***SentencesDataset***, clasa pentru instanțierea unui dataset folosind smart batching, adică fiecare batch este integrat în cea mai lungă secvență, în loc să îmbinăm toate secvențele până la lungimea maximă.
- ***DataLoader***, clasa ce încarcă practic datele în model după batch size
- ***CosineSimilarityLoss***, ce calculează similaritatea cosinus cât și mean squared error-ul dintre embeddings.

Pentru partea de citire a datasetului `dev`, avem nevoie tot de 3 clase:

- ***SentencesDataset***
- ***DataLoader***
- ***EmbeddingsSimilarityEvaluator***, clasă ce compune distanțele euclidenă și manhattan, cât și similaritatea cosinus alături de producția punctelor în maniera Pearson cât și Spearman. Aceasta lucrează, după cum îi spune și numele, ca un evaluator.

Urmează partea de antrenare. Antrenamentul acesta este unul destul de costisitor deoarece arhitectura sistemului meu nu deține un GPU NVIDIA pentru a putea folosi CUDA, drept pentru care antrenarea se va face folosind CPU-ul. Spre exemplu, prima epocă a durat 54 de minute și 32 de secunde pentru a putea fi procesată, cu o viteză de aproape 9 secunde/iterație. Dacă aș fi deținut un GPU NVIDIA, timpul ar fi fost mult mai mic. CPU-urile sunt concepute pentru sarcini de lucru mai generale. În schimb, GPU-urile sunt mai puțin flexibile, cu toate acestea, GPU-urile sunt concepute pentru a calcula în paralel aceleași instrucțiuni. Rețele neuronale spre exemplu, sunt structurate într-o manieră foarte uniformă, astfel încât la fiecare strat al rețelei, mii de neuroni artificiali identici efectuează același calcul. Prin urmare, structura unei rețele se potrivește destul de bine cu tipurile de calcul pe care le poate efectua un GPU eficient. GPU-urile au avantaje suplimentare față de procesoare, printre care se numără mai multe unități de calcul și având o lățime de bandă mai mare de recuperat din memorie. Mai mult, în aplicațiile care necesită procesare capacitățile specifice graficii GPU pot fi exploatate la accelerarea în continuare a calculelor.


```
Epoch: 0%| | 0/4 [00:00<?, ?it/s]
Iteration: 100%| | 360/360 [54:32<00:00, 9.09s/it]
2020-06-10 17:26:50 - Evaluation th
```

Figura 15 Timp/Epoch

```
2020-06-19 15:55:05 - Evaluation the model on dataset after epoch 1:
Convert Evaluating: 100%|
2020-06-19 15:55:05 - Cosine-Similarity : Pearson: 0.7738 Spearman: 0.7713
2020-06-19 15:55:05 - Manhattan-Distance: Pearson: 0.7151 Spearman: 0.7272
2020-06-19 15:55:05 - Euclidean-Distance: Pearson: 0.7165 Spearman: 0.7290
2020-06-19 15:55:05 - Dot-Product-Similarity: Pearson: 0.5971 Spearman: 0.6522
```

Figura 16 Rezultate dupa Epoch 1

La final, după ce modelul este antrenat, acesta este salvat pentru a fi verificat. Verificarea se face folosind câteva propoziții text, în care este căutată asemănarea cea mai mare cu propoziția query. Scorul reprezintă cât de apropiată este propoziția de query-ul dat.

```
Query: Someone in a gorilla costume is playing a set of drums.

Top 5 most similar sentences in corpus:
A monkey is playing drums. (Score: 0.5769)
A cheetah is running behind its prey. (Score: 0.3210)
The girl is carrying a baby. (Score: 0.0901)
A man is riding a white horse on an enclosed ground. (Score: 0.0748)
A man is riding a horse. (Score: 0.0590)
```

Figura 17 Exemplu de verificare GloVe+ Tf-idf

Query-ul poate de asemenea să fie un set de propoziții. Deoarece acestea cât și corpus-ul vor fi datele acestui embedder, acestea trebuie encodate în maniera necesară embedder-ului, adică, a modelului nostru.

```
epoch,steps,cosine_pearson,cosine_spearman,euclidean_pearson,euclidean_spearman,manhattan_pearson,manhattan_spearman,dot_pearson,dot_spearman
0,-1,0.7589698632516429,0.755437142275222,0.7152818194330652,0.7270503430617471,0.7137590162241165,0.7250472225851278,0.5730172517061082,0.618552339761041
1,-1,0.7737955022125085,0.7712835410354405,0.7165240429547061,0.7290373211364441,0.715089608904945,0.727173375495445,0.5971158719775763,0.6521828915890638
2,-1,0.777721995997499,0.7746118602044008,0.7166657322533949,0.7296148977718867,0.7152954640240173,0.7281065831954567,0.5969194233383474,0.6558694261780349
3,-1,0.7791201590615741,0.7761649855433914,0.7167663970583155,0.7299695252744317,0.7153622596416177,0.7284698441783951,0.5962554178582397,0.6560700778396171
4,-1,0.7800685458749077,0.7770349533853296,0.716815613041613,0.7301431361665331,0.7154271663289999,0.7286824640957765,0.5960693142296775,0.6564046525284268
5,-1,0.7809324949631188,0.7777819385361048,0.716931492393804,0.730393918131316,0.7155448956327074,0.7289167993871724,0.59580277375197,0.6565722683032083
6,-1,0.7814095983739112,0.7782066201382125,0.7170100654421079,0.7305924009914586,0.7156333036811388,0.7291306363904131,0.5961090592774531,0.6569450805339984
7,-1,0.7817030410867448,0.7785352991568498,0.7170845948702506,0.7308118892588643,0.7157010552485169,0.7292774322475755,0.5959538947915524,0.6568937400567372
8,-1,0.7818739185347428,0.7788091899161522,0.7171040726521436,0.7308438822468009,0.7157324108358742,0.7293032188093762,0.5958802784181647,0.65693929524852
9,-1,0.7819671079486407,0.7788504071604676,0.7171126989546228,0.7308445163504877,0.7157462962557571,0.7293074750511528,0.5960737231588104,0.65717416018033
```

Figura 18 Rezultate Finale GloVe+ TfIdf

După cele 10 epoci, modelul a ajuns la un scor relativ bun și anume Pearson: 0.6961 Spearman: 0.6705.

În final, folosind o variabilă n care poate fi modificată, în funcție de această variabilă se pot găsi primele n propoziții bazate pe similaritatea dintre acestea folosind similaritatea cosinus.

3.2.2 BERT

BERT este antrenat pe 4 epoch-uri (deoarece conține un număr mare de date deja, fiind pre-antrenat de cei de la Google), în 16 batch-uri.

Deoarece folosim o versiune de pe HuggingFace, BERT încarcă configurările sale înainte de a fi antrenat.

```
2020-06-19 16:32:08 - loading configuration file https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-config.json from cache at /home/yungraz/.cache/torch/transformers/4dad8251492946e18ac39298fcfe91b89d378fee258efe9521476438fe8ca185.7156163d5fd189c3010baca0775ffce230789d7fa2a42ef516483e4ca884517
2020-06-19 16:32:08 - Model config BertConfig {
  "architecture": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
2020-06-19 16:32:09 - loading weights file https://cdn.huggingface.co/bert-base-uncased-pytorch_model.bin from cache at /home/yungraz/.cache/torch/transformers/f2ee78bdd635b758cc8a12352586868bef80e47401abed4fcc3832421e7338b.36ca03ab34a1a5d5fa7bc3083d55c4fa658fced72862eeebc06ce58d0e9a157
2020-06-19 16:32:14 - loading configuration file https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-config.json from cache at /home/yungraz/.cache/torch/transformers/4dad8251492946e18ac39298fcfe91b89d378fee258efe9521476438fe8ca185.7156163d5fd189c3010baca0775ffce230789d7fa2a42ef516483e4ca884517
```

Figura 19 BERT Pre-antrenat integrat în aplicație

Acesta va fi de asemenea antrenat pe STS Benchmark, folosind modelul bert-base-uncased pentru maparea token-urilor pe embeddings.

Pașii sunt relativ asemănători, doar ca BERT nu necesită o rețea neuronală pe care să se antreneze. Drept pentru care, după realizarea polling-ului vectorului, la fel ca la GloVe, putem începe antrenarea pe setul STS Benchmark.

```
Epoch: 0% | 0/4 [00:00<?, ?it /s]
Iteration: 100% | 360/360 [54:32<00:00, 9.09s/it] 2020-06-19 17:28:58 - Evaluation the model on dataset after epoch 0:
Epoch: 0% | 0/4 [54:32<?, ?it /s]
Convert Evaluating: 98% | Convert Evaluating: 100% | Convert Evaluating: 99% |
94/94 [03:06<00:00, 1.98s/it]
2020-06-19 17:29:57 - Cosine-Similarity : Pearson: 0.8698 Spearman: 0.8718
Epoch: 0% | 2020-06-19 17:29:57 - Manhattan-Distance: Pearson: 0.8251 Spearman: 0.8342
Epoch: 0% | 2020-06-19 17:29:57 - Euclidean-Distance: Pearson: 0.8254 Spearman: 0.8344
Epoch: 0% | 2020-06-19 17:29:57 - Dot-Product-Similarity: Pearson: 0.8007 Spearman: 0.8121
Epoch: 0% | 2020-06-19 17:29:57 - Save model to output/training_stsbenchmark_bert-base-uncased-2020-06-19_16-32-07
Epoch: 0% | 2020-06-19 17:29:57 - Configuration saved in output/training_stsbenchmark_bert-base-uncased-2020-06-19_16-32-07/0_Transformer/config.json
Epoch: 0% |
```

Figura 20 BERT antrenat după Epoch 0

Din câte putem observa, un Epoch dureaza 54 de minute ca să fie antrenat folosind CPU. Rezultatele după Epoch-ul 0 arata un scor mult mai mare, Cosine-Similarity : Pearson: 0.8698 Spearman: 0.8718.

Urmează să vedem rezultatul final al antrenării.

```
2020-06-19 23:39:29 - Evaluation the model on dataset:
Convert Evaluating: 100% |
2020-06-19 23:42:08 - Cosine-Similarity : Pearson: 0.8521 Spearman: 0.8490
2020-06-19 23:42:08 - Manhattan-Distance: Pearson: 0.8420 Spearman: 0.8424
2020-06-19 23:42:08 - Euclidean-Distance: Pearson: 0.8425 Spearman: 0.8430
2020-06-19 23:42:08 - Dot-Product-Similarity: Pearson: 0.7794 Spearman: 0.7709
```

Figura 21 Rezultat Evaluare BERT

```
epoch,steps,cosine_pearson,cosine_spearman,euclidean_pearson,euclidean_spearman,manhattan_pearson,manhattan_spearman,dot_pearson,dot_spearman
0,-1,0.8685321413787577,0.872132340302,0.8410081751617736,0.8490152104696184,0.8407030414811448,0.8487871854012239,0.8069387835672942,0.8121255408835498
1,-1,0.8751775034267616,0.875987866094412,0.8442374615918211,0.8499014281630572,0.8436060598677604,0.849363393983925,0.8155984206702879,0.8211630595759539
2,-1,0.8765647667071712,0.877392701583477,0.8510332826435901,0.8551558143514427,0.8501936780210392,0.8545920879837615,0.8275067875206555,0.8313944136118095
3,-1,0.8767863840175204,0.8775288031008897,0.8519863195498999,0.8562288140736639,0.8511653375339198,0.8556429250078877,0.8268457347062396,0.8307094166978047
```

Figura 22 Rezultat BERT dupa cele 4 Epochs

De aici deducem ca, din punct de vedere al performanței antrenării, BERT este mai precis în rezultat decât GloVe+TfIdf. Acest lucru face modelul să fie mai bine optimizat pentru task-ul de recunoastere a similarității.

Pentru verificarea noastră, folosind aceleași date ca la GloVe+TfIdf, BERT a obținut

```
Query: Someone in a gorilla costume is playing a set of drums.  
  
Top 5 most similar sentences in corpus:  
A monkey is playing drums. (Score: 0.7620)  
A cheetah is running behind its prey. (Score: 0.1501)  
A man is eating a piece of bread. (Score: 0.1184)  
A man is riding a horse. (Score: 0.1068)  
A man is eating food. (Score: 0.0895)
```

Figura 23 Exemplu verificare BERT

Deja observăm o îmbunătățire, BERT a prezis cea mai similara propoziție cu un scor mai mare cu aproximativ 0.20 decât ceea ce a furnizat GloVe. (Rezultat BERT cea mai similara propoziție: 0.7620, Rezultat GloVe cea mai similara propoziție: 0.5769).

3.2.3 RoBERTa

RoBERTa va fi antrenată similar cu BERT, pe 4 epochs, în 16 batch-uri. Folosim de asemenea un model pre-antrenat de RoBERTa, furnizat de către librăria HuggingFace. Însă observăm la încărcarea acestuia că dispune o mărime a vocabularului de aproximativ 50.265 de tokens, față de BERT care avea doar 30.522 de tokens. Răman de văzut rezultatele.

```
2020-06-20 15:59:03 - Model config RobertaConfig {
  "architectures": [
    "RobertaForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "roberta",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 1,
  "type_vocab_size": 1,
  "vocab_size": 50265
}
```

Figura 24 RoBERTa configuration

Acest model va fi de asemenea antrenat folosind datasetul STSBenchmark, asemanător cu BERT.

După primul epoch obținem rezultatele:

```
2020-06-20 16:54:52 - Evaluation the model on dataset after epoch 0:
Convert Evaluating: 100% | 94/94 [03:02<00:00, 1.94s/it]
2020-06-20 16:57:55 - Cosine-Similarity : Pearson: 0.8658 Spearman: 0.8685
2020-06-20 16:57:55 - Manhattan-Distance: Pearson: 0.8552 Spearman: 0.8566
2020-06-20 16:57:55 - Euclidean-Distance: Pearson: 0.8559 Spearman: 0.8573
2020-06-20 16:57:55 - Dot-Product-Similarity: Pearson: 0.7911 Spearman: 0.7974
2020-06-20 16:57:55 - Save model to output/training_stsbenchmark_roberta-base-2020-06-20_15-58-09
2020-06-20 16:57:55 - Configuration saved in output/training_stsbenchmark_roberta-base-2020-06-20_15-58-09/0_Transformer/config.json
```

Figura 25 RoBERTa antrenat dupa Epoch 0

După cum putem observa, RoBERTa furnizează un scor mai precis după primul epoch decât BERT. (Vezi figura 17)

Antrenarea epoch-ului 0 a fost, din nou, destul de costisitoare, realizându-se în 54 de minute și 49 de secunde cu un scor de 9.14 secunde per iterație.

În final, RoBERTa a obținut un rezultat aproape identic cu BERT după parcurgerea datasetului de testare:

```
Convert Evaluating: 100%|
2020-06-20 19:53:38 - Cosine-Similarity :      Pearson: 0.8523 Spearman: 0.8493
2020-06-20 19:53:38 - Manhattan-Distance:      Pearson: 0.8346 Spearman: 0.8350
2020-06-20 19:53:38 - Euclidean-Distance:      Pearson: 0.8350 Spearman: 0.8354
2020-06-20 19:53:38 - Dot-Product-Similarity:  Pearson: 0.8110 Spearman: 0.8046
(env) yungraz@DESKTOP-EBYMK06:~/Licenta$
```

Figura 26 Rezultat RoBERTa

Pentru verificarea, RoBERTa a clasificat:

```
Query: Someone in a gorilla costume is playing a set of drums.

Top 5 most similar sentences in corpus:
A monkey is playing drums. (Score: 0.7230)
A man is riding a white horse on an enclosed ground. (Score: 0.1626)
A man is riding a horse. (Score: 0.1356)
A cheetah is running behind its prey. (Score: 0.0952)
Two men pushed carts through the woods. (Score: 0.0230)
```

Figura 27 Exemplu verificare RoBERTa.

Pentru verificare, pipeline-ul ar fii:

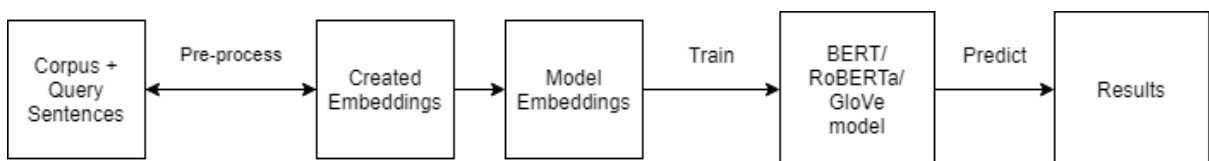


Figura 28 Pipeline Verificare Model

Pe scurt, modelul preia corpus-ul și textul căutat din acest corpus, după care acesta este pre procesat în embeddings cunoscute de BERT. Modelul antrenat este apoi pus să gasească, bazat pe metrica similarității cosinus cele mai apropiate 5 rezultate.

Pentru antrenare și evaluare, pipeline-ul ar fii:

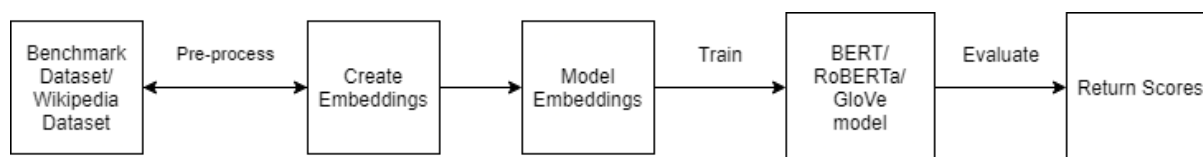


Figura 29 Pipeline Antrenare Model + Evaluare

Pe scurt, datasetul este citit, tokenizarea lui este realizată, iar apoi i se aplica mean pooling pentru a lua un singur sentence vector. Datasetul este convertit în un obiect DataLoader, antrenat și stabilită pierderea modelului. Evaluatorul este inițializat, și apoi începe antrenarea modelului, aceasta fiind bineînțeles configurată cu 10% din data pentru încălzire. Apoi modelul este salvat în folderul de output, iar evaluarea sa este făcuta pe partea de test a datasetului.

3.3 Testarea finala

Testarea finală a fost făcută, folosind câteva documente pentru corpus, și un document referința query, care a fost interogat pentru plagiarism, iterativ prin tot corpusul. Pentru o ușurință mai mare a prelucrării, am denumit documentele relativ asemănătoare cu a/b și numărul documentului.

Pentru ușurința afișării, vom parcurge indexul fiecărui document, îl vom afișa. Vom printa similaritățile alături de restul documentelor, cat si de acesta ca sa demonstram ca modele antrenează corect si afișează similaritatea 1. Apoi, vor fi afișate documentele după prima propoziție din acestea.

Primul document A a fost despre Popeyes, al doilea document B a fost despre McDonald's, ambele fast food-uri. Al doilea document A a fost despre pruna, iar al doilea document B a fost despre cireșe, ambele fructe. Al treilea document A a fost despre Mona Lisa de Leonardo da Vinci, al treilea document B despre Sărutul de Gustav Klimt. Al patrulea document A a fost despre Știința Calculatoarelor, al patrulea document B a fost despre Învățarea Automată, ambele științe. Al cincilea document A a fost despre DotA2, al cincilea document B a fost despre League of Legends, ambele jocuri. Documentele au fost alese astfel încât să testăm un rezultat așteptat al similarităților.

Am implementat pentru ușurință o funcție utilă pentru împărțirea textului în propoziții cu o dimensiune care să permită ca vectorii să fie procesați în BERT/RoBERTa/GloVe., cât și de încărcare a datelor.

Documentul este astfel prelucrat în embeddings, pentru a putea fi procesat în model și testat rezultatul.

```
(env) yungraz@DESKTOP-EB0YK06:~/Licenta$ /home/yungraz/Licenta/env/bin/python /home/yungraz/Licenta/VerifyBert.py

=====

Document query index: 0

Most similar document in corpus:
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 1.0000)
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 0.7801)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.4205)
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 0.4157)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.4124)

=====

Document query index: 1

Most similar document in corpus:
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 1.0000)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.7801)
Computer science is the study of computation and information. (Score: 0.3589)
League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot Games for Microsoft Windows and macOS. (Score: 0.3442)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.3104)

=====

Document query index: 2

Most similar document in corpus:
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 1.0000)
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 0.8178)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.4205)
The Kiss (in German Der Kuss) is an oil-on-canvas painting with added gold leaf, silver and platinum, by the Austrian Symbolist painter Gustav Klimt. (Score: 0.3071)
The Mona Lisa (; Italian: Monna Lisa [ˈmɔnna ˈliːza] or La Gioconda [la dʒo ˈkɔnda]; French: La Joconde [la ʒɔˈkɔd]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.3036)

=====

Document query index: 3

Most similar document in corpus:
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 1.0000)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.8178)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.4157)
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 0.2975)
The Kiss (in German Der Kuss) is an oil-on-canvas painting with added gold leaf, silver and platinum, by the Austrian Symbolist painter Gustav Klimt. (Score: 0.2839)
```

Figura 30 Rezultat document BERT

BERT dovedește prin acest rezultat că este un model extraordinar pentru context, înregistrând scoruri mari la documentul numărul 2 și 3, unde a dedus o similaritate inclusiv între documentul despre prune, cât și despre McDonalds, pe scurt, mâncarea.

```
(env) yungraz@DESKTOP-EB0YK06:~/Licenta$ /home/yungraz/Licenta/env/bin/python /home/yungraz/Licenta/VerifyGlove.py

=====

Document query index: 0

Most similar document in corpus:
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 1.0000)
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 0.8010)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.3555)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.3537)
Computer science is the study of computation and information. (Score: 0.2500)

=====

Document query index: 1

Most similar document in corpus:
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 1.0000)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.8010)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.4010)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.3949)
Computer science is the study of computation and information. (Score: 0.3852)

=====

Document query index: 2

Most similar document in corpus:
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 1.0000)
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 0.8751)
League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot Games for Microsoft Windows and macOS. (Score: 0.5344)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.4802)
The Mona Lisa (; Italian: Monna Lisa [ˈmɔnna ˈliːza] or La Gioconda [la dʒo ˈkɔnda]; French: La Joconde [la ʒɔˈkɔd]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.4059)

=====

Document query index: 3

Most similar document in corpus:
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 1.0000)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.8751)
League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot Games for Microsoft Windows and macOS. (Score: 0.4767)
The Kiss (in German Der Kuss) is an oil-on-canvas painting with added gold leaf, silver and platinum, by the Austrian Symbolist painter Gustav Klimt. (Score: 0.4600)
The Mona Lisa (; Italian: Monna Lisa [ˈmɔnna ˈliːza] or La Gioconda [la dʒo ˈkɔnda]; French: La Joconde [la ʒɔˈkɔd]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.4178)
```

Figura 31 Rezultat document GloVe

GloVe în schimb, nu vede analogia BERT-ului. Se vede că acesta nu este un model care să rețină contextul atât de bine ca BERT, însă oferă rezultate bune textelor similare. Putem spune că nu este atât de creativ ca BERT. Însă modelul oferă un rezultat bun în legătură cu asemănarea textuală.

```
(env) yungraz@058109-E8-VN000:~/Licenta$ /home/yungraz/Licenta/env/bin/python /home/yungraz/Licenta/VerifyRoBERTa.py

=====

Document query index: 0

Most similar document in corpus:
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 1.0000)
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 0.7952)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.3943)
The Mona Lisa (; Italian: Monna Lisa ['mɔnna 'liːza] or La Gioconda [la dʒo'konda]; French: La Joconde [la ʒɔkɔ̃d]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.3889)
League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot Games for Microsoft Windows and macOS. (Score: 0.3686)

=====

Document query index: 1

Most similar document in corpus:
McDonald's Corporation is an American fast food company, founded in 1940 as a restaurant operated by Richard and Maurice McDonald, in San Bernardino, California, United States. (Score: 1.0000)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.7952)
Computer science is the study of computation and information. (Score: 0.3992)
Machine learning (ML) is the study of computer algorithms that improve automatically through experience. (Score: 0.3797)
Dota 2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve. (Score: 0.3734)

=====

Document query index: 2

Most similar document in corpus:
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 1.0000)
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 0.7908)
The Kiss (in German Der Kuss) is an oil-on-canvas painting with added gold leaf, silver and platinum. by the Austrian Symbolist painter Gustav Klimt. (Score: 0.3843)
The Mona Lisa (; Italian: Monna Lisa ['mɔnna 'liːza] or La Gioconda [la dʒo'konda]; French: La Joconde [la ʒɔkɔ̃d]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.3360)
Popeyes is an American multinational chain of fried chicken fast food restaurants founded in 1972 in New Orleans, Louisiana and headquartered in Miami, Florida. (Score: 0.2502)

=====

Document query index: 3

Most similar document in corpus:
A cherry is the fruit of many plants of the genus Prunus, and is a fleshy drupe (stone fruit). (Score: 1.0000)
A plum is a fruit of the subgenus Prunus of the genus Prunus. (Score: 0.7908)
The Mona Lisa (; Italian: Monna Lisa ['mɔnna 'liːza] or La Gioconda [la dʒo'konda]; French: La Joconde [la ʒɔkɔ̃d]) is a half-length portrait painting by the Italian artist Leonardo da Vinci. (Score: 0.3633)
The Kiss (in German Der Kuss) is an oil-on-canvas painting with added gold leaf, silver and platinum. by the Austrian Symbolist painter Gustav Klimt. (Score: 0.3310)
League of Legends (LoL) is a multiplayer online battle arena video game developed and published by Riot Games for Microsoft Windows and macOS. (Score: 0.3138)
```

Figura 32 Rezultat document RoBERTa

RoBERTa are alte asocieri decât BERT, este de asemenea un model care este solid pe partea de similaritate textuala, atât semantica. Dacă observați rezultatele BERT vs RoBERTa pe primul document, o să vedeți că sunt relativ asemănătoare din punct de vedere al detectării celor mai apropiate documente. Însă, RoBERTa a făcut o asociere interesanta: a favorizat documentul cu Dota ca al treilea din punct de vedere al similarității, cu scorul de 0.39, în timp ce BERT a clasat documentul cu prunele al treilea.

Pe scurt, toate modelele au oferit rezultatul așteptat.

4 TERMENI DE UTILIZARE

4.1 Autorii

Autorul acestui document de licență este studentul Roșu Răzvan – Virgil, coordonat de către coordonatorul științific dl. Conf.dr.ing. Mihăescu Cristian Marian.

5 CONCLUZII

Acest proiect a prezentat crearea acestor modele care reușesc să ducă sarcina la capăt, cu concluziile pe care le-am prezis. GloVe este un model independent de context, cu un vector pentru fiecare cuvânt, combinând toate sensurile diferite în un singur vector. BERT și RoBERTa sunt modele care generează word embeddings diferite pentru un cuvânt, și îi acoperă și contextul acestuia. GloVe nu deține astfel context sensitive embeddings. Metoda de predicție este una bună din punctul de vedere al rezultatelor modelului GloVe + Tf-Idf, însă BERT și RoBERTa au depășit cu mult această model în performanță

Următorii pași ar fi integrarea acestora în un site / aplicație desktop și chiar folosirea lor pentru a depista plagiarismul pe seturi de big data.

Acest proiect m-a ajutat să îmi dezvolt abilitățile de cercetare a unor lucruri noi, precum procesarea limbajului natural, cât și relativ cunoscute deja, precum învățarea automată. De asemenea, simt că mi-am dezvoltat abilitățile de programare în Python, cât și în învățarea automată prin ducerea la capăt al acestui proiect.

6 BIBLIOGRAFIE

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Algorithms And Data Structures, Dumitru Dan Burdescu, Marian Cristian Mihăescu, Editura Academică Greifswald, 2011

NLP: The Essential Guide to Neuro-Linguistic Programming, Tom Dotz, Tom Hoobyar, 2013

The Hundred-Page Machine Learning Book, Andriy Burkov, 2019

Python Cookbook: Recipes for Mastering Python 3, David Beazley, Brian K. Jones, 2011

7 REFERINȚE WEB

- [1] – Învățare automata - https://ro.wikipedia.org/wiki/%C3%8Env%C4%83%C8%9Bare_automat%C4%83
- [2] – Procesarea limbajului natural - <https://sites.google.com/site/andreiscutelnicu/nlp>
- [3] – Python - <http://purepython.eaudeweb.ro/wiki/Cursuri/Introducere.html>
- [4] – Numpy - <https://en.wikipedia.org/wiki/NumPy>
- [5] - Tensorflow- <https://www.guru99.com/what-is-tensorflow.html>
- [6]-Transformer- [https://en.wikipedia.org/wiki/Transformer_\(machine_learning_model\)](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))
<https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>
- [7]-BERT-<https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>
<https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>
https://huggingface.co/transformers/model_doc/bert.html
<https://www.kaggle.com/abhinand05/bert-for-humans-tutorial-baseline>
<https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03>
- [8]-GloVe- [https://en.wikipedia.org/wiki/GloVe_\(machine_learning\)](https://en.wikipedia.org/wiki/GloVe_(machine_learning))
- [9]-TF-IDF-<https://ro.wikipedia.org/wiki/Tf%E2%80%93idf>
- [10]- Similaritatea cosinus - <https://www.machinelearningplus.com/nlp/cosine-similarity/>
https://en.wikipedia.org/wiki/Cosine_similarity
<https://www.machinelearningplus.com/nlp/cosine-similarity/>
- [11]- RoBERTa- <https://arxiv.org/abs/1907.11692>
- [12]-Metrici de evaluare - <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/>
- [13]-Epoch-<https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>

8 CODUL SURSĂ

GloveTFIDF.py

```
1. from torch.utils.data import DataLoader
2. import math
3. from sentence_transformers import SentenceTransformer, SentencesDataset, LoggingHandler, losses, models
4. from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
5. from sentence_transformers.readers import STSBenchmarkDataReader
6. import logging
7. from datetime import datetime
8. import sys
9.
10.
11. # print debug information to stdout
12. logging.basicConfig(format='%(asctime)s - %(message)s',
13.                     datefmt='%Y-%m-%d %H:%M:%S',
14.                     level=logging.INFO,
15.                     handlers=[LoggingHandler()])
16.
17. # Read the dataset
18. batch_size = 32
19. sts_reader = STSBenchmarkDataReader('Data/stsbenchmark')
20. model_save_path = 'output/training_tf-idf_word_embeddings-'
    '+datetime.now().strftime("%Y-%m-%d_%H-%M-%S")'
21.
22.
23.
24. # Map tokens to traditional word embeddings like GloVe
25. word_embedding_model = models.WordEmbeddings.from_text_file('glove.6B.300d.txt.gz')
26.
27. # Weight word embeddings using Inverse-Document-Frequency (IDF) values.
28. # For each word in the vocab of the tokenizer, we must specify a weight value.
29. # The word embedding is then multiplied by this value
30. vocab = word_embedding_model.tokenizer.get_vocab()
31. word_weights = {}
32. lines = open('wikipedia_doc_frequencies.txt').readlines()
33. num_docs = int(lines[0])
34. for line in lines[1:]:
35.     word, freq = line.strip().split("\t")
36.     word_weights[word] = math.log(num_docs/int(freq))
37.
38. # Words in the vocab that are not in the doc_frequencies file get a frequency of 1
39. unknown_word_weight = math.log(num_docs/1)
40.
41. # Initialize the WordWeights model. This model must be between the WordEmbeddings and the Pooling model
42. word_weights = models.WordWeights(vocab=vocab, word_weights=word_weights, unknown_word_weight=unknown_word_weight)
43.
44.
45. # Apply mean pooling to get one fixed sized sentence vector
46. pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(),
47.                                pooling_mode_mean_tokens=True,
48.                                pooling_mode_cls_token=False,
49.                                pooling_mode_max_tokens=False)
50.
```

```

51. # Add two trainable feed-forward networks (DAN)
52. sent_embeddings_dimension = pooling_model.get_sentence_embedding_dimension()
53. dan1 = models.Dense(in_features=sent_embeddings_dimension, out_features=sent_embeddings_dimension)
54. dan2 = models.Dense(in_features=sent_embeddings_dimension, out_features=sent_embeddings_dimension)
55.
56. model = SentenceTransformer(modules=[word_embedding_model, word_weights, pooling_model, dan1, dan2])
57.
58.
59. # Convert the dataset to a DataLoader ready for training
60. logging.info("Read STSBenchmark train dataset")
61. train_data = SentencesDataset(sts_reader.get_examples('sts-train.csv'), model=model)
62. train_dataloader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
63. train_loss = losses.CosineSimilarityLoss(model=model)
64.
65. logging.info("Read STSBenchmark dev dataset")
66. dev_data = SentencesDataset(examples=sts_reader.get_examples('sts-dev.csv'), model=model)
67. dev_dataloader = DataLoader(dev_data, shuffle=False, batch_size=batch_size)
68. evaluator = EmbeddingSimilarityEvaluator(dev_dataloader)
69.
70. # Configure the training
71. num_epochs = 10
72. warmup_steps = math.ceil(len(train_data) * num_epochs / batch_size * 0.1) #10% of train data for warm-up
73. logging.info("Warmup-steps: {}".format(warmup_steps))
74.
75. # Train the model
76. model.fit(train_objectives=[(train_dataloader, train_loss)],
77.           evaluator=evaluator,
78.           epochs=num_epochs,
79.           warmup_steps=warmup_steps,
80.           output_path=model_save_path
81.           )
82.
83. # Load the stored model and evaluate its performance on STS benchmark dataset
84.
85. model = SentenceTransformer(model_save_path)
86. test_data = SentencesDataset(examples=sts_reader.get_examples("sts-test.csv"), model=model)
87. test_dataloader = DataLoader(test_data, shuffle=False, batch_size=batch_size)
88. evaluator = EmbeddingSimilarityEvaluator(test_dataloader)
89.
90. model.evaluate(evaluator)

```

RoBERTa.py

```

1. from torch.utils.data import DataLoader
2. import math
3. from sentence_transformers import SentenceTransformer, SentencesDataset, LoggingHandler, losses, models
4. from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
5. from sentence_transformers.readers import STSBenchmarkDataReader
6. import logging
7. from datetime import datetime
8. import sys
9.
10. # print debug information to stdout

```

```

11. logging.basicConfig(format='%(asctime)s - %(message)s',
12.                      datefmt='%Y-%m-%d %H:%M:%S',
13.                      level=logging.INFO,
14.                      handlers=[LoggingHandler()])
15. ##### /print debug information to stdout
16.
17. model_name = 'roberta-base'
18.
19. # Read the dataset
20. train_batch_size = 16
21. num_epochs = 4
22. model_save_path = 'output/training_stsbenchmark_'+model_name.replace("/", "-")+ '-'
    '+datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
23. sts_reader = STSBenchmarkDataReader('Data/stsbenchmark', normalize_scores=True)
24.
25. # Use Huggingface/transformers model (like BERT, RoBERTa, XLNet, XLM-
    R) for mapping tokens to embeddings
26. word_embedding_model = models.Transformer(model_name)
27.
28. # Apply mean pooling to get one fixed sized sentence vector
29. pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(),

30.                                pooling_mode_mean_tokens=True,
31.                                pooling_mode_cls_token=False,
32.                                pooling_mode_max_tokens=False)
33.
34. model = SentenceTransformer(modules=[word_embedding_model, pooling_model])
35.
36. # Convert the dataset to a DataLoader ready for training
37. logging.info("Read STSBenchmark train dataset")
38. train_data = SentencesDataset(sts_reader.get_examples('sts-train.csv'), model)
39. train_dataloader = DataLoader(train_data, shuffle=True, batch_size=train_batch_size
    )
40. train_loss = losses.CosineSimilarityLoss(model=model)
41.
42.
43. logging.info("Read STSBenchmark dev dataset")
44. dev_data = SentencesDataset(examples=sts_reader.get_examples('sts-
    dev.csv'), model=model)
45. dev_dataloader = DataLoader(dev_data, shuffle=False, batch_size=train_batch_size)
46. evaluator = EmbeddingSimilarityEvaluator(dev_dataloader)
47.
48.
49. # Configure the training. We skip evaluation in this example
50. warmup_steps = math.ceil(len(train_data)*num_epochs/train_batch_size*0.1) #10% of t
    rain data for warm-up
51. logging.info("Warmup-steps: {}".format(warmup_steps))
52.
53.
54. # Train the model
55. model.fit(train_objectives=[(train_dataloader, train_loss)],
56.           evaluator=evaluator,
57.           epochs=num_epochs,
58.           evaluation_steps=1000,
59.           warmup_steps=warmup_steps,
60.           output_path=model_save_path)
61.
62.
63. # Load the stored model and evaluate its performance on STS benchmark dataset
64.
65. model = SentenceTransformer(model_save_path)
66. test_data = SentencesDataset(examples=sts_reader.get_examples("sts-
    test.csv"), model=model)

```



```

67. test_dataloader = DataLoader(test_data, shuffle=False, batch_size=train_batch_size)
68. evaluator = EmbeddingSimilarityEvaluator(test_dataloader)
69. model.evaluate(evaluator)

```

BERT.py

```

1. from torch.utils.data import DataLoader
2. import math
3. from sentence_transformers import SentenceTransformer, SentencesDataset, LoggingHandler, losses, models
4. from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator
5. from sentence_transformers.readers import STSBenchmarkDataReader
6. import logging
7. from datetime import datetime
8. import sys
9.
10. #print debug information to stdout
11. logging.basicConfig(format='%(asctime)s - %(message)s',
12.                     datefmt='%Y-%m-%d %H:%M:%S',
13.                     level=logging.INFO,
14.                     handlers=[LoggingHandler()])
15.
16. model_name = 'bert-base-uncased'
17.
18. # Read the dataset
19. train_batch_size = 16
20. num_epochs = 4
21. model_save_path = 'output/training_stsbenchmark_'+model_name.replace("/", "-")+ '-'
22. +datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
23. sts_reader = STSBenchmarkDataReader('Data/stsbenchmark', normalize_scores=True)
24.
25. # Use Huggingface/transformers model (like BERT, RoBERTa, XLNet, XLM-R) for mapping tokens to embeddings
26. word_embedding_model = models.Transformer(model_name)
27.
28. # Apply mean pooling to get one fixed sized sentence vector
29. pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(),
30.                                pooling_mode_mean_tokens=True,
31.                                pooling_mode_cls_token=False,
32.                                pooling_mode_max_tokens=False)
33.
34. model = SentenceTransformer(modules=[word_embedding_model, pooling_model])
35.
36. # Convert the dataset to a DataLoader ready for training
37. logging.info("Read STSBenchmark train dataset")
38. train_data = SentencesDataset(sts_reader.get_examples('sts-train.csv'), model)
39. train_dataloader = DataLoader(train_data, shuffle=True, batch_size=train_batch_size)
40.
41. train_loss = losses.CosineSimilarityLoss(model=model)
42.
43. logging.info("Read STSBenchmark dev dataset")
44. dev_data = SentencesDataset(sts_reader.get_examples('sts-dev.csv'), model=model)
45. dev_dataloader = DataLoader(dev_data, shuffle=False, batch_size=train_batch_size)
46. evaluator = EmbeddingSimilarityEvaluator(dev_dataloader)
47.

```

```

48. # Configure the training. We skip evaluation in this example
49. warmup_steps = math.ceil(len(train_data)*num_epochs/train_batch_size*0.1) #10% of t
    rain data for warm-up
50. logging.info("Warmup-steps: {}".format(warmup_steps))
51.
52.
53. # Train the model
54. model.fit(train_objectives=[(train_dataloader, train_loss)],
55.           evaluator=evaluator,
56.           epochs=num_epochs,
57.           evaluation_steps=1000,
58.           warmup_steps=warmup_steps,
59.           output_path=model_save_path)
60.
61. # Load the stored model and evaluate its performance on STS benchmark dataset
62.
63. model = SentenceTransformer(model_save_path)
64. test_data = SentencesDataset(examples=sts_reader.get_examples("sts-
    test.csv"), model=model)
65. test_dataloader = DataLoader(test_data, shuffle=False, batch_size=train_batch_size)
66. evaluator = EmbeddingSimilarityEvaluator(test_dataloader)
67. model.evaluate(evaluator)

```

VerifyBERT.py

```

1. from sentence_transformers import SentenceTransformer
2. import scipy.spatial
3. import numpy as np
4. import PrepareData
5.
6. embedder = SentenceTransformer('output/training_stsbenchmark_bert-base-uncased-
    2020-06-19_19-51-26')
7.
8. corpus = PrepareData.load_data()
9.
10. # Corpus
11. corpus_embeddings = []
12. for document in corpus:
13.     sentences_embeddings = embedder.encode(document)
14.     sentences_embeddings = np.array(sentences_embeddings)
15.     document_embedding = np.mean(sentences_embeddings, axis = 0)
16.     corpus_embeddings.append(document_embedding)
17.
18.
19. # Query sentences:
20. #
21. #similarity_matrix = []
22. #for first_doc in corpus_embeddings:
23. #    similarity_vector = []
24. #    for second_doc in corpus_embeddings:
25. #        similarity_vector.append(1 - scipy.spatial.distance.cosine(first_doc, seco
    nd_doc))
26. #    similarity_matrix.append(similarity_vector)
27. #
28. #similarity_matrix = np.array(similarity_matrix)
29. #print(similarity_matrix)
30.
31.
32. # Find the closest 5 documents of the corpus for each query sentence based on cosin
    e similarity
33. closest_n = 5

```

```

34. index = 0
35. for query, query_embedding in zip(corpus, corpus_embeddings):
36.     distances = scipy.spatial.distance.cdist([query_embedding], corpus_embeddings,
        "cosine")[0]
37.
38.     results = zip(range(len(distances)), distances)
39.     results = sorted(results, key=lambda x: x[1])
40.
41.     print("\n\n===== \n\n")
42.     print("Document query index:", index)
43.     print("\nMost similar document in corpus:")
44.
45.     for idx, distance in results[0:closest_n]:
46.
47.         print(corpus[idx][0], "(Score: %.4f)" % (1-distance))
48.
49.     index = index + 1

```

VerifyRoBERTa.py

```

1. from sentence_transformers import SentenceTransformer
2. import scipy.spatial
3. import numpy as np
4. import PrepareData
5.
6. embedder = SentenceTransformer('output/training_stsbenchmark_roberta-base-2020-06-
    20_15-58-09')
7.
8. corpus = PrepareData.load_data()
9.
10. # Corpus with example sentences
11. corpus_embeddings = []
12. for document in corpus:
13.     sentences_embeddings = embedder.encode(document)
14.     sentences_embeddings = np.array(sentences_embeddings)
15.     document_embedding = np.mean(sentences_embeddings, axis = 0)
16.     corpus_embeddings.append(document_embedding)
17.
18.
19. # Query sentences:
20. #
21. #similarity_matrix = []
22. #for first_doc in corpus_embeddings:
23. #    similarity_vector = []
24. #    for second_doc in corpus_embeddings:
25. #        similarity_vector.append(1 - scipy.spatial.distance.cosine(first_doc, seco
        nd_doc))
26. #    similarity_matrix.append(similarity_vector)
27. #
28. #similarity_matrix = np.array(similarity_matrix)
29. #print(similarity_matrix)
30.
31.
32.
33.
34. # Find the closest 5 documents of the corpus for each query sentence based on cosin
    e similarity
35. closest_n = 5
36. index = 0
37. for query, query_embedding in zip(corpus, corpus_embeddings):
38.     distances = scipy.spatial.distance.cdist([query_embedding], corpus_embeddings,
        "cosine")[0]

```

```

39.
40.     results = zip(range(len(distances)), distances)
41.     results = sorted(results, key=lambda x: x[1])
42.
43.     print("\n\n=====\n\n")
44.     print("Document query index:", index)
45.     print("\nMost similar document in corpus:")
46.
47.     for idx, distance in results[0:closest_n]:
48.
49.         print(corpus[idx][0], "(Score: %.4f)" % (1-distance))
50.
51.     index = index + 1

```

VerifyGloVe.py

```

1. from sentence_transformers import SentenceTransformer
2. import scipy.spatial
3. import numpy as np
4. import PrepareData
5.
6. embedder = SentenceTransformer('output/training_tf-idf_word_embeddings-2020-06-
   19_15-54-05')
7.
8. corpus = PrepareData.load_data()
9.
10. # Corpus with example sentences
11. corpus_embeddings = []
12. for document in corpus:
13.     sentences_embeddings = embedder.encode(document)
14.     sentences_embeddings = np.array(sentences_embeddings)
15.     document_embedding = np.mean(sentences_embeddings, axis = 0)
16.     corpus_embeddings.append(document_embedding)
17.
18.
19. # Query sentences:
20. #
21. #similarity_matrix = []
22. #for first_doc in corpus_embeddings:
23. #    similarity_vector = []
24. #    for second_doc in corpus_embeddings:
25. #        similarity_vector.append(1 - scipy.spatial.distance.cosine(first_doc, seco
   nd_doc))
26. #    similarity_matrix.append(similarity_vector)
27. #
28. #similarity_matrix = np.array(similarity_matrix)
29. #print(similarity_matrix)
30.
31.
32.
33.
34. # Find the closest 5 documents of the corpus for each query sentence based on cosin
   e similarity
35. closest_n = 5
36. index = 0
37. for query, query_embedding in zip(corpus, corpus_embeddings):
38.     distances = scipy.spatial.distance.cdist([query_embedding], corpus_embeddings,
   "cosine")[0]
39.
40.     results = zip(range(len(distances)), distances)
41.     results = sorted(results, key=lambda x: x[1])
42.

```

```

43.     print("\n\n===== \n\n")
44.     print("Document query index:", index)
45.     print("\nMost similar document in corpus:")
46.
47.     for idx, distance in results[0:closest_n]:
48.
49.         print(corpus[idx][0], "(Score: %.4f)" % (1-distance))
50.
51.     index = index + 1

```

Utils.py

```

1. import re
2. def split_text_max_words(lines, max_words):
3.     sentences = []
4.
5.     for line in lines:
6.         splits = re.finditer("\. [A-Z]|\.\n", line)
7.         last_split_idx = 0
8.         for split in splits:
9.             sentence = line[last_split_idx: split.start() + 1]
10.            if(len(sentence.split(" ")) >= 5):
11.                sentences.append(sentence)
12.                last_split_idx = split.start() + 2
13.
14.    sentenceSplittedByMaxWords = []
15.
16.    for sentence in sentences:
17.        words = sentence.split(" ")
18.
19.        for i in range(len(words) // max_words):
20.            begin = i*max_words
21.            end = (i+1)*max_words
22.            sentenceSplittedByMaxWords.append(" ".join(words[begin : end]))
23.
24.        if(len(words) % max_words != 0):
25.            begin = (len(words) // max_words)*max_words
26.            end = (len(words) // max_words)*max_words + (len(words) % max_words)
27.            sentenceSplittedByMaxWords.append(" ".join(words[begin : end]))
28.
29.    return sentenceSplittedByMaxWords

```

PrepareData.py

```

1. import os
2. import Utils
3.
4. def load_data():
5.
6.     files = []
7.     documents = []
8.     path = 'Data/Plagiarism Documents';
9.     for r, d, f in os.walk(path):
10.        for file in f:
11.            if '.txt' in file:
12.                files.append(path+'/'+file)
13.
14.
15.    for path in files:
16.        file = open(path, 'r')
17.        lines = file.readlines()

```

```
18.         documents.append(Utils.split_text_max_words(lines, 128))
19.
20.     return documents
21.
22.
23.
```

9 CD / DVD

Repository GIT - <https://github.com/YungRAW/PlagiarismChecker>

