Proiect la Inteligență Artificială

Bran Andreea, Grupa 244

Aprilie, 2020

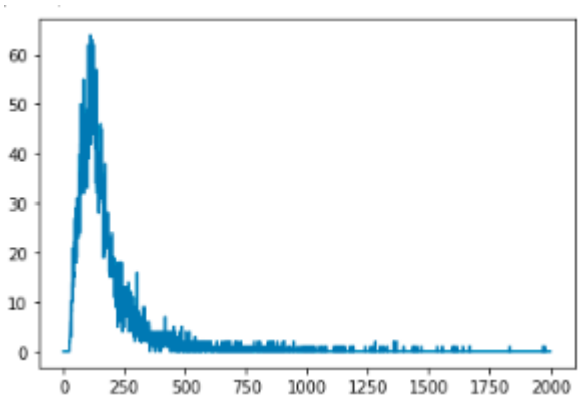
Cuprins

1 Rezumat

În cadrul acestei documentații vom trata problematica identificării de dialect. În primul rând, se va prezenta analiza datelor, urmând să exemplificăm două mari clase de abordări: la nivel de caracter, folosind rețele convoluționale(Character Level CNN), și la nivel de cuvânt, folosind Tf-idf și clasificatori clasici. Vom prezenta si câteva statistici ale rezultatelor, comparând metodele.

2 Analiza datelor

Setul de date de antrenare,(respectiv validare), vine cu 7757 de texte (respectiv 2656).



În figura de mai sus se poate observa că datele sunt de lungimi foarte variate, cea mai mare parte fiind concentrată în intervalul(0,700].
De asemenea, datele sunt formate dintr-un alfabet de 60 de caractere (spațiu inclus).

3 Abordări

3.1 Rețele convoluționale la nivel de caracter

Precizie: 64.75%
F1 macro:64.09%

Această abordare este inspirată din 2 articole [1] [2]. Ambele au în prim-plan distincția dintre dialecte, primul axându-se pe diferențierea dintre graiul românesc și cel moldovenesc, iar cel de al doilea pe distincția dintre

dialectele arabe. Aceste 2 lucrări tratează problema operand la nivel de caracter, folosind rețele convoluționale.

În interesul nostru este mai ales lucrarea [1], întrucat vom încerca să implementăm această rețea și să o adaptăm la cerința noastră.

3.1.1 Reprezentarea datelor

Primul pas este trierea setului de date. Întrucat textele au lungimi extrem de variate, alegerea a fost de a împărți datele de intrare în texte de un număr fixat N de caractere, urmand ca predicțiile să se facă pe baza primelor N caractere din textele de validare, respectiv de testare.

Întrucat un număr mare de texte este concentrat în intervalul (0,700], preferința va fi de a împărți textele în secvențe de nu mai mult de N=300 de caractere, urmând ca secvențelor mai scurte să li se adauge bordare pentru a ajunge la aceasta lungime. Această alegere a fost făcută experimental. (Alte lungimi alese au fost 150, 200, 250, 400, 700, 1000, însă rezultatele au fost mai puțin promițătoare).

Acest aspect va ajuta parțial (foarte puțin) și la compensarea numărului mic de date.

Folosind aceaste lungimi, vom crea un alfabet pe baza textelor de intrare, pe care îl vom reprezenta sub forma unui dicționar cu chei și valori unice. Astfel, fiecare text va fi encifrat prin înlocuirea literelor cu valoarea unica a acestora din dicționar.

Primul bloc de embedding va fi bazat pe aceasta reprezentare, construind o matrice de dimeniuni 62x61 (lungimea alfabetului+2, respectiv +1), care pe prima linie are doar 0, iar urmatoarele 61 linii formează matricea I_{61}. Există o linie și o coloană în plus pentru caracterele pe care le vom găsi și care nu se află în alfabet, iar prima linie corespunde padding-ului.

3.1.2 Arhitectura rețelei

Arhitectura rețelei se va păstra în mare parte, mergand pe abordarea cu blocuri de tip Squeeze and Excitation. Justificarea acestei alegeri se face prin simpla observație experimentală a comportamentului rețelei, care este mult mai "stabilă" în ajustarea ponderilor atunci când sunt adăugate aceste blocuri(oscilațiile de acuratețe și recall sunt mai graduale, acuratețea

pe testare "ține mai bine pasul" cu acuratețea de pe antrenare).

Rețeaua va fi construită din 3 blocuri convoluționale fiecare urmat de 3 blocuri de tip Squeeze and Excitation. Pentru blocurile convoluționale, activările vor fi de tip LeakyReLu; mărimea filtrelor rămâne de 7 pentru primele 2 blocuri și 3 pentru ultimul. Pentru blocurile de tip Squeeze and Excitation specificațiile se patrează: o activare de tip ReLu si una de tip sigmoid, operația finala de average-pooling va fi tot globala, iar numărul de filtre pentru stratul intermediar va fi de 2. Pentru LeakyReLu păstrăm alpha= $5 \cdot 10^{-5}$, deoarece și experimental aceasta valoare s-a potrivit. Am încercat înlocuirea activării LeakyReLu cu cea de tip Swish, însă rețeaua făcea overfit mult mai ușor, iar la partea de predicții tendința era de a prezice mereu aceeași clasă pentru toate textele, timp de mai multe epoci. Numărul de filtre ales pentru blocurile convoluționale este 128; din dorința de a reduce numărul de parametri, dar și din observații experimentale, și 64 de filtre dau rezultate asemanatoare.

Blocurile interconectate vor păstra o activare de tip ReLu, iar numărul de filtre va fi de 1024. Alte valori de filtre încercate au mai fost si 256 și 512, dintre care mai promițător a fost 512, în combinație cu 64 de filtre pentru blocurile convoluționale și batch-uri de antrenare de 64.

Blocul de ieșire are activare de tip softmax. Alte alegeri pentru acesta au fost funcțiile de tip sigmoid și Elliot, dintre care și Elliot a dat rezultate de acurateță asemănătoare, existând însă diferențe în cadrul recall-ului.

Antrenarea se va face timp de 10-12 epoci(după acest număr, rețeaua face doar overfit, iar acuratețea pe testare stagnează în jurul pragului de 64 sau scade%), in batch-uri de 128, cu optimizer de tip Adam și rata de învățare $5 \cdot 10^{-4}$, și clasificare de tip binary crossentropy.

În plus, vom alege un prag diferit pentru probabilitățile predicțiilor. Vom considera clasa 1 orice element pentru care probabilitatea de a aparține clasei 1 este mai mare de 0.6. Deși diferența de acurateță este extrem de mica, scorul F1 este cu 1-2% mai mare pentru acest prag.

3.1.3 Detalii de implementare ale modelului

Modelul este implementat folosind modelul **Model** din **Keras**, iar straturile se găsesc în pachetul **keras.layers**.

Vom folosi straturi de tip **Dense** pentru blocurile interconectate,**Conv1D** pentru straturile convoluționale, **GlobalAveragePooling1D** pentru operația de pooling, și **LeakyRelu** pentru activările cu același nume.

Pentru input și embedding se vor folosi Input cu parametru 300 si Embedding, care va avea dimensiunea matricei one-hot.

Din păcate, numărul epocilor, care în medie nu ar trebui să fie mai mult de 10, depinde și de o serie de factori aleatori, cum ar fi seed-urile; există situații în care 10-12 epoci sunt insuficiente și metricile rețelei cresc mult mai greu (pot fi necesare 15-17 epoci pentru acelasi rezultate). Rezultatele prezentate aici sunt obtinute pentru cel mai bun seed aleator. Cu toate acestea, pe cazul mediu, acuratețea se păstrează în intervalul 62-64%, iar un seed bun s-a dovedit a fi 12345 pentru blocurile complet interconectate, în cazul în care alegem numărul de filtre 128:1024. Pentru cazul cu 64:512, acesta dă rezultate aproximativ la fel de bune.

3.2 TfidfVectorizer și clasificatoarele clasice

Această abordare este o alternativă a versiunii de clasificare BagOfWords, care stoca frecvențele cuvintelor. Pentru toți clasificatorii clasici folosiți în această secțiune, preferința va fi de a folosi această codificare.

3.2.1 Reprezentarea datelor

În reprezentarea datelor cu **TfidfVectorizer**, vom păstra toate caracterele întâlnite, cu excepția caracterelor blank sau de tip spațiu. De asemenea, nu vom transforma literele mari în litere mici, deoarece acestea reprezintă codificări relevante, și vom păstra toate caracterele speciale din același motiv. Vom alege, de asemenea, să păstrăm toate cuvintele, indiferent de lungime, folosind n-grame de (1,1), iar normalizarea va fi de tip L2.

3.2.2 Mașini cu vectori suport(SVM)

Precizie: 70.01%
F1 macro: 69.99%

Pentru a găsi parametri ideali ai unui SVM în combinație cu **TfidfVectorizer**, vom folosi **GridSearchCV** din biblioteca **sklearn**.

Opțiunile pentru paramtrul C alese sunt $\{1, 10, 100\}$, pentru functia Gamma $\{1, 0.1, 0.001\}$, iar pentru funcția de kernel luăm in considerare kernelul liniar,

rbf și cel de tip polinomial.

Cele mai bune rezultate s-au obținut pentru C=10, Gamma=1 și kernel de tip polinomial.

3.2.3 RandomForest

Precizie:66.94%
F1 macro: 66.91%

Pentru **RandomForest**, vom căuta parametri folosind **GridSearchCV**. Pentru adâncimea maximă am ales mulțimea $\{50, 110, 700, 1000\}$, $\{200, 500, 700\}$, iar pentru **minimum samples split** $\{50, 70, 100\}$. Cele mai bune rezultate au fost obținute pentru adâncimea maximă 1000, 200 de estimatori și **minimum samples split** de 100.

3.2.4 ComplementNB

Precizie: 71.87%
F1 macro: 71.84%

Drept clasificator se folosește **ComplementNB** din biblioteca **sklearn**. Drept parametri am luat în considerare opțiunile $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.001\}$ pentru care am înregistrat cea mai bună precizie la 0.05. În căutarea parametrilor am folosit **GridSearchCV**.

4 Statistici

Pentru abordarea rețelelor neuronale, matricea de confuzie este următoarea:

Matrix	0	1
0	680	621
1	315	1040

Pentru abordarea care folosește drept clasificator SVC, matricea de confuzie este următoarea:

Matrix	0	1
0	820	481
1	312	1043

Pentru abordarea care folosește drept clasificator RandomForest, matricea de confuzie este următoarea:

Matrix	0	1
0	853	448
1	430	925

Pentru abordarea care folosește drept clasificator ComplementNB, matricea de confuzie este următoarea:

Matrix	0	1
0	912	389
1	358	997

Se poate observa că în acest caz, clasa mai dificil de prezis este '0' și că atât modelul bazat pe CNN, cât și cel bazat pe SVM tind să clasifice mai multe elemente de tip '0' în clasa '1'. În cadrul abordărilor ce folosesc clasificatorul de tip ComplementNB, respectiv RandomForest, se poate observa în schimb că numărul de elemente catalogate greșit din fiecare clasă este destul de echilibrat.

5 Concluzii

Este interesant de observat că, pentru acest set de date, abordările de "shallow" au rezultate mai bune. Un motiv ar putea fi numărul mic de date, dar și distribuția lor: datele sunt de lungimi foarte variate, iar dacă textele scurte fac parte din tweet-uri și cele lungi din articole, asta ar putea fi iar o problema.

6 Bibliografie

[1] Radu T. Ionescu Andrei M. Butnaru. Moroco: The moldavian and romanian dialectal corpus, 2019.

[2] Yann LeCun Xiang Zhang, Junbo Zhao. Character-level convolutional networks for text classification, 2015.