Romanian sub-dialect identification

Voinea Andrei

Facultatea de Matematică și Informatică

1. Descrierea task-ului

În această lucrare, vom lucra la un model, care va face diferența dintre dialectul românesc și cel moldovenesc. Datele pe care vom antrena modelul sunt compuse din 7757 de propoziții. Setul de validare este construit din 2656 de propoziții, iar cel de testare este format din 2623 de astfel de propoziții. Labelurile folosite pentru fiecare propoziție vor fi 0 pentru dialect moldovenesc, respectiv 1 pentru dialect românesc.

2. Preprocesarea cuvintelor

Propozițiile sunt criptate, însă spațiul dintre cuvinte nu și-a pierdut însemnitatea, așadar indicând delimitarea și separarea cuvintelor. Prin urmare, ne putem folosi de abordarea „Bag of Words”, care presupune crearea unui vocabular cu toate cuvintele găsite în propozițiile folosite pentru antrenare, pe care îl vom folosi ulterior la crearea unei matrice care va măsura numărul aparițiilor fiecărui cuvânt în fiecare propoziție. Astfel, spre exemplu, $M_{ij} = x$ înseamnă că propoziția ***i*** conține cuvântul ***j*** de ***x*** ori (deci i = numărul propoziției curente ; j = numărul asociat cuvântului de pe poziția respectivă ; x = un număr natural).

Pentru această abordare, vom folosi modulul CountVectorizer, cu parametrul „token_pattern=’\S+’”, pentru a selecta orice caracter care **NU** este blanc. Tot cu ajutorul modulului CountVectorizer, putem selecta n-grame de cuvinte sau caractere. Pe lângă acestea, alți parametri folositori pe care îi vom modifica cu scopul de a obține un scor cât mai bun vor fi „lowercase” (*True*), „ngram_range” (*(1,1)*), „analyzer” (*word*),

„max_df” (*1.0*), „min_df” (*1*). Așadar, pe datele de antrenare vom avea următoarele numere de trăsături (s-au ales doar câteva exemple):

	Lowercase = True	***Lowercase = False***
(1, 1) ngrame de cuvinte	32,301	32,874
(1, 2) ngrame de cuvinte	253,567	254,217
(5, 5) ngrame de caractere	380,970	434,735

Cum influențează acești parametri modelul se va discuta pe parcursul documentației.

Cu toate acestea, vom folosi **tf-idf** (term frequency-inverse document frequency), care este o metodă de a reflecta importanța unui cuvânt într-o propoziție în raport cu restul documentului (celelalte propoziții). Deci, un cuvânt care apare în fiecare propoziție va avea ponderea mai mică decât un cuvânt care apare într-o singură propoziție. **TF(w)** = (numărul de apariții a cuvântului w într-o propoziție) / (numărul total de cuvinte din propoziția respectivă)

IDF(w) = $\log_e$ (numărul total de propoziții / numărul de propoziții în care se găsește cuvântul w)

Spre exemplu:

```
propozitie = ["Ana are mere", "Vasile are mere"]
tfidf = TfidfVectorizer()
propozitie = tfidf.fit_transform(propozitie)
print(tfidf.get_feature_names())
# ['ana', 'are', 'mere', 'vasile']
print(propozitie.toarray())
# [[0.70490949 0.50154891 0.50154891 0.        ]
#  [0.         0.50154891 0.50154891 0.70490949]]
```

Observăm că numărul cuvintelor / n-gramelor rămâne același ca la CountVectorizer, dar se va schimba matricea cu care vom lucra.

Parametrii „*max_df*” și „*min_df*” sunt folosiți pentru a evita cuvintele care apar de prea multe ori, respectiv de prea puține ori. Dacă setăm parametrul

„*max_df*” la 0.3 (apare în mai mult de 30% din propoziții), vom observa că se va elimina un cuvânt.

Normalizarea datelor joacă un rol foarte important în cazul nostru. Prin normalizare, înțelegem schimbarea valorilor numerice, astfel încât lungimea textelor folosite nu afectează ponderea cuvintelor. Alt exemplu util, independent de taskul pe care îl avem, ar fi un set de date cu două trăsături diferite: vârstă (0 – 100+) și salariu (1000 – 10000+), care sunt în două intervale total diferite, dar modelul nu ar trebui să ia decizii uitându-se doar la numărul mai mare, oferindu-i mai multă importanță. **Normalizarea se va face doar** pe datele scoase cu CountVectorizer, deoarece pe **tf-idf** are un **efect neglijabil**, din cauza formulei pe care metoda o folosește.

Alte metode de preprocesare pe care le-am fi putut aborda în plus dacă cuvintele noastre nu erau criptate erau:

- Eliminarea numerelor / semnelor de punctuație / cuvintelor comune (stop words)
- Eliminarea diacriticelor (de dezbătut, deoarece dialectul moldovenesc folosește „î” în mijlocul propozițiilor, în timp ce în dialectul românesc, aceeași literă se folosește doar la începutul cuvintelor)
- Lematizarea cuvintelor (i.e. aducerea cuvântului la forma de bază)

3. Modele abordate

Modelele folosite pentru acest task, cu cele din urmă (ultimele două) fiind mai eficiente, au fost:

a. k-nearest neighbors (KNN)
b. logistic regression
c. support vector classification
d. naive bayes multinomial

Pe parcurs se va face diferența și dintre CountVectorizer (atât a datelor înainte și după normalizare) și TfidfVectorizer, cât și dintre parametrii folosiți la modele.

Parametrii care nu vor apărea în următoarele tabele vor fi: *lowercase* = False, *max_df* = 0.3, *min_df* = 1, *ngram_range* = (1, 2), *analyzer* = 'word'. Aceștia vor fi neschimbați pentru fiecare tabel.

a) **k-nearest neighbors (KNN)**

	CountVectorizer		CountVectorizer Date normalizate		TfidfVectorizer	
weights / n-neighbors	uniform	distance	uniform	distance	uniform	distance
1	51.28	51.28	62.12	62.12	65.88	65.88
3	53.68	53.72	60.35	60.39	64.15	64.19
5	51.84	51.88	60.01	60.09	64.60	64.64
10	52.10	52.99	58.69	60.95	65.09	67.62
20	51.65	51.69	59.82	60.88	64.94	66.94

Au fost testate și datele tfidf normalizate, dar acuratețea rămâne la fel în orice situație, astfel, demonstrându-se că normalizarea datelor tfidf este neglijabilă.

b) **logistic regression**

	CountVectorizer			CountVectorizer Date normalizate			TfidfVectorizer		
C / max_iter	500	2500	5000	500	2500	5000	500	2500	5000
1	68.03	68.03	68.03	67.73	67.73	67.73	67.92	67.92	67.92
10	67.77	67.77	67.77	68.93	68.93	68.93	69.16	69.16	69.16
100	67.84	67.84	67.84	68.63	68.63	68.63	69.35	69.35	69.35

Au fost testate și datele tfidf normalizate, dar acuratețea rămâne la fel în orice situație, astfel, demonstrându-se că normalizarea datelor tfidf este neglijabilă.

Datorită faptului că datele tfidf normalizate au dat aceeași acuratețe (sau neglijabilă, i.e. +0.000001) pe ultimele două modele, de acum încolo modelele nu vor mai prezice și pe acestea.

c) **support vector classification**

	CountVectorizer		CountVectorizer Date normalizate		TfidfVectorizer	
kernel / C	rbf	linear	rbf	linear	rbf	linear
1	62.87	67.09	68.10	68.14	68.03	68.82
10	66.82	67.09	69.23	68.52	69.91	69.39
100	66.82	67.09	69.23	68.52	69.91	69.39

d) **naive bayes multinomial**

alpha	CountVectorizer	CountVectorizer Date normalizate	TfidfVectorizer
1	71.00	68.90	70.21
0.1	72.36	72.43	73.30
0.01	71.53	73.71	72.25
0.001	70.82	72.40	71.31
0.0001	70.82	71.79	71.00

Se pot observa diferențele dintre preprocesări, cât și diferențele dintre parametrii folosiți de modele. O primă concluzie ar fi că metoda TfidfVectorizer dă mai bine în 3/4 modele, iar în cel de al patrulea model este la o diferență de 0.41 de metoda care dă cel mai bine. O altă observație ar fi că dintre cele două soluții care folosesc CountVectorizer, cea unde datele sunt normalizate are rata de succes mai mare.

4. Parametri folosiți la preprocesare

Deoarece, cele mai bune rezultate le-am avut pe Naive Bayes Multinomial, îl vom folosi pe acesta (cu alpha = 0.1) pentru testarea parametrilor folosiți de TfidfVectorizer. Tabelul următor a fost **ordonat descrescător după acuratețe**.

maxDF	minDF	lowercase	ngramRange	analyzer	acuratețe
0.3	1	False	(1, 2)	word	73.30
0.7	1	False	(1, 2)	word	73.19
0.3	1	False	(1, 3)	word	73.04
0.3	1	True	(1, 2)	word	72.92
0.3	1	True	(1, 3)	word	72.74
0.3	1	True	(1, 1)	word	71.91
0.3	1	False	(1, 1)	word	71.68
0.3	2	False	(1, 2)	word	71.19
0.7	5	False	(6, 6)	char	70.93
0.3	2	False	(6, 6)	char	70.48
0.7	2	False	(6, 6)	char	70.48
0.3	1	False	(6, 6)	char	69.12
0.3	1	False	(5, 5)	char	69.05
0.3	1	False	(5, 8)	char	68.78
0.7	10	False	(6, 6)	char	68.59
0.3	1	False	(7, 7)	char	67.69
0.3	1	False	(8, 8)	char	66.11
0.3	1	False	(2, 2)	word	65.21

Ce putem extrage din acest tabel:

- păstrarea cuvintelor cu majuscule (lowercase = False) ne crește acuratețea
- n-gramele pe cuvinte oferă rezultate mai bune decât cele pe caractere pentru Naive Bayes
- eliminarea cuvântului care apare în mai mult de 30% din documente crește acuratețea

5. Modelul ales și metodele abordate

După testarea mai multor modele / preprocesări, am ajuns să aleg să folosesc două abordări diferite cu Naive Bayes Multinomial. Prima abordare are următorii parametri pentru:

- TfidfVectorizer: max_df = 0.3, min_df = 1, lowercare = False, token_pattern = '\S+', ngram_range = (1, 2), analyzer = 'word'
- MultinomialNB: alpha = 0.1

Matricea de confuzie:

Clasa prezisă / Clasa actuală	0	1
0	926	375
1	334	1021

Positive = 1; Negative = 0

TP = 1021

FP = 375

FN = 334

Precizie = TP / TP + FP = 1021 / 1021 + 375 = 1021 / 1396 = 0.73

Recall = TP / TP + FN = 1021 / 1021 + 334 = 1021 / 1355 = 0.75

F_1 = 2 * P * R / (P + R) = 1.095 / 1.48 = 0.7398 (rezultatul din program va da diferit față de acesta, din cauză că aici folosim doar două zecimale)

A doua abordare este combinarea a mai multor clasificatori pentru a extrage niște propoziții din datele de testare și de a le folosi ulterior ca date de antrenare. Soluția este predispusă la a face greșeli, dar numărul de date pe care îl folosim este mic rezultând implicit că și numărul greșelilor va fi mic, așadar ne putem asuma acest risc. Motivul pentru care nu ar merge ar fi în următorul caz: cazul în care toți clasificatorii prezic un test

ca fiind de label 0 (respectiv 1), acesta fiind de 1 (respectiv 0), iar pe urmă antrenând modelul cu date greșite, astfel încurându-l mai departe.

Clasificatori pe care i-am folosit au fost cei 4 descriși anterior, iar condiția de trecere pentru o propoziție de testare să fie folosită și la antrenare a fost ca toți clasificatorii să o clasifice cu același label (deci 4 de 0 sau 4 de 1). Avem două seturi de parametri folosiți pentru clasificatori. Naive Bayes și KNN vor folosi ngrame pe cuvinte, iar logistic regresssion și SVC vor folosi ngrame pe caractere (acuratețe mai bună, i.e. puțin peste 0.70 la ambele). Pe urmă am folosit, Naive Bayesul cu parametrii de la cealaltă abordare pe noile date de antrenare.

Matricea de confuzie:

Clasa prezisă / Clasa actuală	0	1
0	904	397
1	339	1016

Positive = 1; Negative = 0

TP = 1016

FP = 397

FN = 339

Precizie = TP / TP + FP = 1016 / 1016 + 397 = 1016 / 1413 ≈ 0.72 (0.719)

Recall = TP / TP + FN = 1016 / 1016 + 339 = 1016 / 1355 ≈ 0.75 (0.749)

F_1 = 2 * P * R / (P + R) = 1.08 / 1.47 = 0.7346 (rezultatul din program va da diferit față de acesta, din cauză că aici folosim doar două zecimale)