Alien Language Classification

Documentatie

- *Obiectiv:* Diferentierea intre limbile extratereste vorbite pe planete diferite

- *Overview*: Cele doua clasificatoare utilizate in submisiile finale sunt SGDClassifier si MultinomialNB cu modificarile aferente aduse parametrilor. Am obtinut cel mai mult pe submisia cu SGDClassifier.

1. Preluarea si formatarea datelor

Pentru a utiliza clasificatorii mentionati anterior, datele (initial in format text) au trebuit sa fie modificate pentru a corespunde tipului parametrilor acestora, mai exact parametrilor functiei fit(X,Y) prin care antrenam modelul.

In acest sens, X reprezinta datele de antrenare, iar in cazul clasificarii limbajului extraterestru consta in textul limbii regasit. Pe baza acestuia, se asociaza (in cazul antrenarii) ori se atribuie (in cazul predictiei) o eticheta corespunzatoare limbii de care apartine. La preluarea datelor, am alcatuit o lista in care am ignorat id-urile sample-urilor, pastrand doar textul limbii extraterestre.

Clasificatorii cer pentru X tipul {array-like, sparse matrix} cu shape-ul (n_samples, n_features). Este asadar necesar sa extragem feature-urile. In acest scop am folosit cate un vectorizer : in primul caz, CountVectorizer, iar in al doilea, TfidVectorizer, ambele importate din sklearn.feature_extraction.text. Acestea preiau textul si creeaza o matrice de “tokens” in functie de aparitiile cuvintelor. Spre deosebire de CountVectorizer, TfidVectorizer creeaza o matrice de TD-IDF features (Term-

Frequency Inverse Document Frequency) , masurand si luand in considerare relevanta cuvintelor.

Pentru submisiile finale, X a fost reprezentat de o lista rezultata din concatenarea datelor de antrenare cu cele de validare. Pentru submisiile in care am studiat evolutia acuratetii, X a fost reprezentat de datele de antrenament. In submisiile anterioare, antrenam intai pe datele de train, afisam studiul predictiei pe cele de validare, iar apoi antrenam din nou pe cele de validare.

Y reprezinta setul de etichete care vor fi asociate datelor. Initial, am retinut etichetele intr-o lista, insa clasificatorii cer formatul ndarray. In acest sens, am importat numpy pentru a putea transforma lista initiala.

Am verificat cu ajutorul unor afisari daca valorile de shape sunt valide pentru X si Y, respectiv X sa aiba drept prima valoare in urma aplicarii functiei shape n_samples, iar Y asemenea, intrucat numarul etichetelor trebuie sa corespunda numarului sample-urilor.

2. MultinomialNB

Prima mea abordare in competitie a fost utilizarea unui clasificator de tip Bayes naiv. Am ales MultinomialNB intrucat este indicat pentru features care reprezinta counts (numarari), iar in urma vectorizarii efectuate asupra datelor, aceasta era si situatia cazului tratat. MultinomialNB prezinta urmatoare avantaje:

cost computational redus, trateaza eficient seturi mari de date, se descurca cu problemele cu mai multe clase (multiclass) si in cele de clasificare a textului.

Initial nu am modificat niciun parametru, ci modelat cu MultinomialNB() simplu, in urma vectorizarii cu TfidVectorizer().

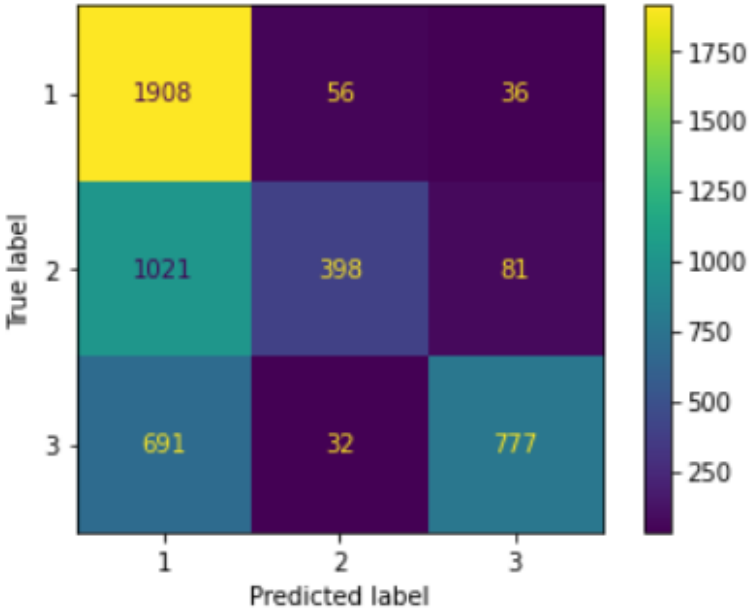
In general, pentru a masura performanta am aplicat, in urma antrenarii strict pe setul de train:

- MultinomialNB.score(date_validare, etichete_validare)
- f1_score(predictia_obtinuta, etichete_validare, average='None')

(parametrul average=None : pentru a afisa scorul pentru fiecare clasa)

Am obtinut:

- 0.6166
- [0.67900356, 0.40080564, 0.64912281]



Am incarcat aceasta sursa ca prima submisie si obtinut 0.56552 (Public Score), 0.56279 (Private Score). Intrucat clasificatorul se descurca bine pe testele oferite de platforma, am experimentat cu valorile parametrul de smoothing (netezire) a datelor : alpha. (Ceilalti parametrii, fit_prior, class_prior faceau referinta la probabilitati ale claselor specificate/invatate anterior si nu am considerat ca este cazul).

Pentru diferite valori ale lui alpha, am obtinut:

Alpha value	score	f1_score
15	0.435	[0.586 0.031 0.1828]
10	0.4426	[0.5898 0.0366 0.2188]
5	0.477	[0.6065 0.093 0.342]
1(default)	0.6166	[0.679 0.4008 0.64912]
0.5	0.664	[0.703 0.536 0.70077]

0.1	0.6778	[0.693 0.61 0.72]
0.01	0.6578	[0.67 0.59 0.70]

Am redus zecimalele unor valori din tabel, pastrandu-le pe cele suficiente pentru relevanta. In urma acestor teste am dedus ca obtin o performanta mai buna pentru un alpha mai mic, dar nu prea mic. In acest caz am descoperit un peak in intervalul [0.1 , 0.2].

Pentru a imbunatati modelul, am testat valorile optime gasite pentru parametrul alpha utilizand, de data aceasta, CountVectorizer. Pentru alpha = 0.02, am obtinut:

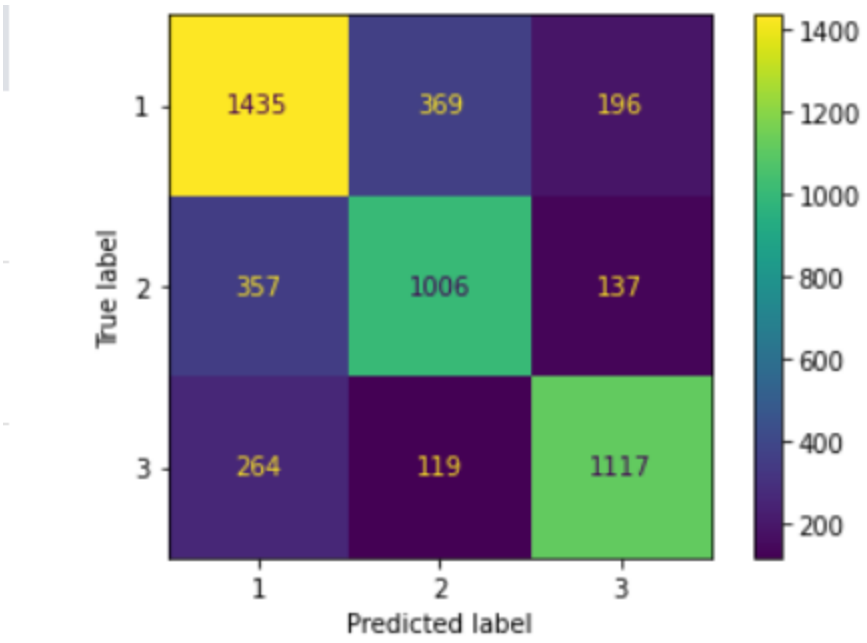
- score: 0.6746
- f1-score: [0.67820575 0.62300645 0.721111149]

Observam crestere fata de valorile obtinute cu TfidfVectorizer incluse in tabel. Am decis sa utilizez acest vectorizer in continuare.

Ultima aditie pe care am adus-o acestui model, respectiv vectorizer, a fost implementarea n-gramelor (intrucat parametrul analyzer='word' by default, n-grame pe cuvinte).

Pentru CountVectorizer(ngram_range=(1,2)) am obtinut:

- score: 0.7062
- f1-score: [0.70436893 0.66463208 0.75059707]



Aceasta este prima submisie finala, iar pe Kaggle a obtinut 0.67301 (Public Score), 0.65914 (Private Score).

Pentru alte valori ale n-gramelor am obtinut:

ngram_range	score	f1-score
(1,3)	0.704	[0.6955 0.664 0.7568]
(1,4)	0.702	[0.6949 0.6622 0.7525]
(2,3)	0.677	[0.6673 0.6372 0.7314]
(4,10)	0.5857	[0.5711 0.5711 0.5750]
(2,10)	0.6684	[0.6605 0.6311 0.7173]
(10,20)	0.4802	[0.6053 0.3831 0.0727]
(20,100)	0.4536	[0.5941 0.2922 0.0158]

Am dedus ca performanta este buna pentru valori mai mici ale n-gramelor.

Consider ca acest range al n-gramelor este datorat faptului ca sunt realizate pe cuvinte, analyzer-ul default fiind ‘word’. Spre exemplu, pentru range-ul (1,2), aplicat insa ‘char’ pentru parametrul analyzer al vectorizerului, am obtinut:

- score: 0.5716
- f1-score: [0.57276 0.56514 0.57570317]

Insa pentru un range mai mare, (1,10) si analyzer = ‘char’:

- score: 0.7568
- f1-score: [0.75476 0.7333 0.78302863]

Pentru ngram_range=(1,15) obtinem:

- score: 0.7428
- f1-score: [0.73885 0.71698113 0.77430436]

Iar pentru ngram_range(1,13):

- score: 0.746
- f1-score: [0.742366 0.72074 0.77636]

Nu am obtinut un score sau f1-score cu analyzer=’word’ si ngram_range (optim pe baza testelor efectuate) pe masura celui obtinut cu analyzer=’char’ si ngram_range optim (pe baza testelor efectuate). Cred ca acest lucru se datoreaza faptului ca limba extraterestra nu functioneaza ca o limba normala in care se repeta adesea cuvintele de baza. Am ajuns la aceasta concluzie la finalul competitiei, inainte sa trimit ultima submisie si am aplicat aceasta idee asupra celui de al doilea model. Consider ca este totusi o comparatie interesanta intre cele doua modele dezvoltate.

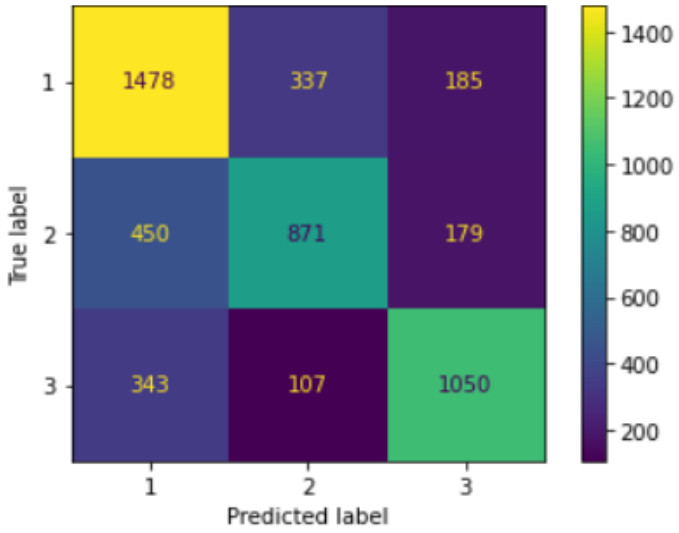
3. SGDClassifier

A doua abordare o reprezinta utilizarea SGDClassifier regasit in sklearn.linear_model. Dupa abordarea initiala in care am optat pentru un clasificator care ofera, in general, valori favorabile, am cautat clasificatori folositi adesea pentru lucrul cu text si language classification, iar SGDClassifier este unul dintre acestia. L-am utilizat cu parametrul loss=’hinge’ (default), iar conform

sklearn, se comporta ca un SVM Linear (l-am considerat o optiune buna, avand in vedere faptul ca doream sa delimitez datele in functie de criteriul apartenentei la limba). In plus, conform documentatiei de pe sklearn pentru Stochastic Gradient Descent, clasificatorul opereaza bine pentru seturi mari de date (inclusiv sparse data). SGD-ul este un algoritm de optimizare al modelelor, iar in urma unor testelor pe SGDClassifier() si LinearSVC(), am obtinut valori mai bune pentru SGDClassifier():

SGDClassifier(max_iter=1000000)

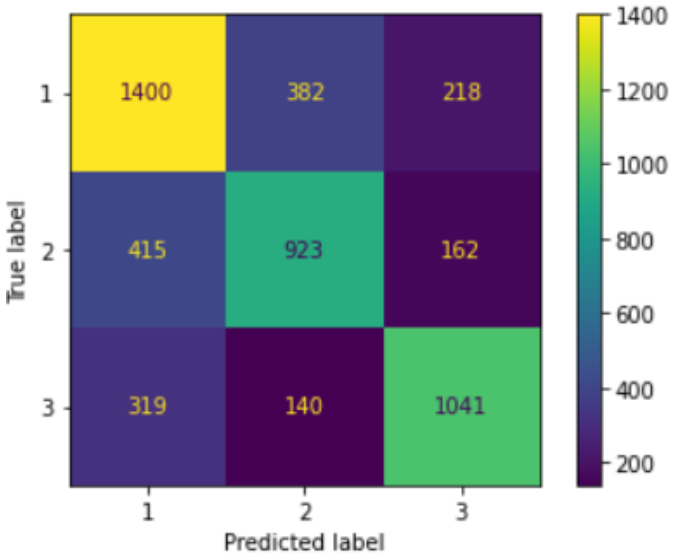
- score: 0.6798
- f1_score: [0.69210958 0.61882771 0.72065889]




```
[0.69210958 0.61882771 0.72065889]
```

LinearSVC(max_iter = 10000)

- score: 0.6728
- f1_score: [0.67731011 0.62682513 0.7127696]



Am decis sa imbunatatesc modelul cu SGDClassifier. Pe parcursul competiei, am optinut pe submisiile care utilizau SGDClassifier:

Submisia 4

SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, random_state=None,max_iter=100000, tol=None

TfidfVectorizer()

Submisia 6

SGDClassifier(loss='hinge',max_iter=10000, tol=1e-3)

TfidfVectorizer(ngram_range=(1, 5)

Submisia 9

SGDClassifier(max_iter=1000000)

TfidfVectorizer(ngram_range=(1,10))

Submisia 10

SGDClassifier(max_iter=1000000)

TfidfVectorizer(ngram_range=(1,17))

Submisia 13 (A doua submisie finala)

SGDClassifier(max_iter=10000)

TfidfVectorizer(ngram_range=(1,13), analyzer='char')

Submisie	Public Score	Private Score
4	0.63661	0.63303
6	0.68659	0.67633
9	0.69939	0.73279
10	0.68500	0.70609
13	0.77586	0.75884

Am efectuat mai multe teste pentru a ajunge la ngrame_range = (1,13), astfel:

SGDClassifier(max_iter=10000)

TfidfVectorizer(ngram_range = (1,?), analyzer = 'char')

ngram_range	score	f1_score
(1,10)	0.7562	[0.7576 0.7341 0.7758]
(1,11)	0.7566	[0.7571 0.7361 0.7763]
(1,12)	0.7576	[0.7587 0.7375 0.7756]
(1,13)	0.7578	[0.7594 0.7369 0.7761]
(1,14)	0.7558	[0.7574 0.7376 0.7714]
(1,16)	0.755	[0.7565 0.7403 0.7675]

Am ales cate patru zecimale la f1_score, relevante pentru observatii.

Am efectuat teste si pentru analyzer=’word’ (care au contribuit la concluzia despre performanta mai buna pe char-uri):

ngram_range	score	f1_score
(1,10)	0.6968	[0.6940 0.6755 0.7255]
(1,11)	0.6932	[0.6913 0.6749 0.7173]
(1,12)	0.691	[0.6885 0.6736 0.7153]

4. Submisii cu alte modele

(Public Score / Private Score source: Kaggle)

Submisia 2

LogisticRegression(class_weight="balanced", max_iter=10000)

TfidfVectorizer()

Public Score: 0.66778

Private Score: 0.65698

LogisticRegression a oferit, de asemenea, rezultate favorabile. Algoritmul este bazat pe conceptul de probabilitate, performand pe tipul de probleme binare in care trebuie sa stabilit daca ceva este adevarat sau fals (in cazul acesta, daca un text apartine sau nu unei limbi). Utilizeaza functia sigmoid(logistic function) pentru a returna probabilitatea unei etichete (o valoare inclusa in intervalul [0,1]). Regresia logistica este utilizata, in general, pentru probleme de clasificare.

Submisia 3

RidgeClassifier()

CountVectorizer(analyzer='char', ngram_range=(1,5))

Public Score: 0.63442

Private Score: 0.62924

RidgeClassifier este, in general, mai rapid decat LogisticRegression si performeaza pentru probleme de clasificare (cum este si cea a limbilor extratereste). Pentru TfidfVectorizer(ngram_range=(1,10)), acelasi apel returneaza pentru f1_scor valorile [0.68186323 0.66465984 0.69795609] (apropiate de cele oferite de LogisticRegression). Probabil sursa ar fi putut obtine un punctaj mai bun cu acest vectorizer, parametrii optimizati prin efectuare de teste pentru ngram_range si alpha.