Alien Language Classification
Documentație

Mihai Radu-Ioan
Grupa 232

1. Introducere

Documentaţia prezintă rezultatele şi statisticile obţinute din clasificarea cuvintelor în funcţie de limba extraterestră din care fac parte: Bellerophon (eticheta 1), Ymir (eticheta 2) sau Nanook (eticheta 3).

2. Citirea şi preprocesarea datelor

2.1 Citirea

Având în considerare faptul că fişierele text de intrare aveau coloanele de date separate printr-un caracter tab (\t), am folosit funcţia csv.reader() cu delimitator="\t" pentru a stoca datele în două tablouri unidimensionale, câte unul pentru fiecare coloană.

Am observat faptul că, în fişierele de intrare, caracterele erau de tip unicode characters. Prin urmare, pentru fişierele în care erau stocate cuvintele, am folosit encoding="mbcs". Pentru fişierele ce conțineau doar etichete, am folosit encoding-ul "utf-8".

2.2 Preprocesare

Iniţial am folosit 3 metode de pre-procesare: dicţionar (vector de frecvenţă al cuvintelor), CountVectorizer şi TfidfVectorizer.

Folosind cele 3 metode de preprocesare cu parametrii default în cazul CountVectorizer şi TfidfVectorizer am comparat rezultatele folosind Multinomial,

Gaussian, Bernoulli şi Complement Naive Bayes. (Alegerea clasificatorilor a fost făcută datorită timpului scurt de executare a acestora.) După aproximativ 5 rulări pe valori alpha diferite, am ajuns la concluzia că cele mai bune rezultate se obţin folosind CountVectorizer, lucru ce m-a determinat să îl păstrez şi pentru alţi clasificatori.

Pentru a îmbunătăți performanța CountVectorizer, m-am folosit de parametrii acestuia:

- lowercase = False, având în considerare faptul că aveam caractere unicode, am considerat că, din punct de vedere logic, lowercase nu ar funcționa la fel de bine. Acest lucru mi s-a confirmat prin teste.
- enconding = ”mbcs”, având în considerare experiența din cadrul citirii, am adăugat enconding și în cadrul CountVectorizer.
- binary = True, am considerat faptul că procesarea datelor este mai ușor de făcut în sistem binar, lucru ce mi s-a demonstrat în cadrul testelor.
- analyzer = ”char”, având în considerare faptul că nu erau caractere normale, deci cuvintele puteau avea frecvență mică, am încercat să fac BagOfWords pe caractere în loc de cuvinte, rezultatele crescând semnificativ.
- ngram_range din mulțimea {(1,6), (1,7), (1,8), (3,7)}. Folosind un program cu 2 for-uri care a testat toate tuplurile (n1,n2) cu n2<=n1<=15 și modelele Naive Bayes, am observant că cele mai bune valori se obțin pe aceste 4 intervale.

3. Clasificatori folosiți

3.1 Clasificatori cu rezultate slabe

Inițial, am folosit clasificatori cu care am mai lucrat pe perioada laboratorului, precum Multinomial Naive Bayes, KNeighbors Classifier, Random Forest Classifier sau MLP Classifier.

Dintre aceastea 4 am păstrat pe 2 cele mai bune după acuratețe (MNB și RFC) pentru a continua antrenarea, în timp ce KNeighbors Classifier și MLP Classifier nu au mai fost folosite pentru viitoare antrenări.

Folosind metoda de preprocesare CountVectorizer descrisă mai sus, pe ngram_range = (1,7), Multinomial Naive Bayes a întors o acuratețe de 0.7524 cu

alpha = 1 (default), în timp ce Random Forest Classifier cu 100 estimatori returna 0.724.

În comparație, KneighborsClassifier întorcea următoarea acuratețe pe datele de validare:

n_neighbors	accuracy_score
1	0.463
5	0.4518
10	0.4554
15	0.4506
20	0.4478
25	0.4464
30	0.4366
35	0.4314
40	0.432
45	0.4336
50	0.43

Am observat faptul că acuratețea era departe de cea înregistrată de cele 2 modele de top și că aceasta tindea să scadă la un număr mai mare de vecini.

După, am rulat MLP Classifier pe diferite straturi și număr de iterații, cele mai bune 5 fiind:

layers	max_iter	accuracy score
(5000)	10	0.6904
(1000, 2000)	10	0.69
(1000, 2000)	9	0.6896
(1000, 2000)	15	0.689
(1000, 1000, 20)	100	0.6828

Cum acuratețea nu a trecut de 0.7, am decis să renunț și la acest clasificator.

3.2 Naive Bayes

Folosind CountVectorizer cu preprocesările menționate la paragraful 2, am rulat MultinomialNB, ComplementNB și BernoulliNB pe diferite valori alpha (parametrul aditiv).

În tabelul de mai jos se poate observa acuratețea pe datele de validare a clasificatorilor. Căsuța colorată cu verde reprezintă cel mai bun scor obținut pe datele de intrare, în timp ce căsuțele galbene reprezintă cel mai bun scor pe datele de validare pe acel ngram_range, respectiv faptul că au fost submisii pe platformă.

	(1,6)	**(1,7)**	**(1,8)**	**(3,7)**
Multinomial NB (alpha = 1)	0.7524	0.7524	0.7488	0.751
Multinomial NB (alpha=0.5)	0.76	0.7636	0.763	0.7624
Multinomial NB (alpha=0.01)	0.7666	0.768	0.7662	0.7676
Complement NB (alpha = 1)	0.7594	0.7644	0.7678	0.7642
Complement NB (alpha=0.5)	0.7612	0.7656	0.7682	0.7652
Bernoulli NB (alpha = 1)	0.7162	0.7164	0.7036	0.7138
Bernoulli NB (alpha=0.5)	0.7212	0.7352	0.741	0.7366

Matricele de confuzie pentru aceste 4 submisii sunt:

ngram_range = (1,6)

```
[[1556  285  245]
 [ 262 1097   75]
 [ 182  118 1180]]
```

ngram_range = (1,7)

```
[[1572  299  247]
 [ 256 1088   73]
 [ 172  113 1180]]
```

ngram_range = (1,8)

```
[[1535  264  226]
 [ 274 1085   53]
 [ 191  151 1221]]
```

ngram_range = (3,7)

```
[[1579  297  251]
 [ 261 1086   76]
 [ 160  117 1173]]
```

3.3 Random Forest Classifier

Datorită faptului că plot-urile RFC și NB sunt asemănătoare [1], am știut de la început că RFC ar putea avea rezultate bune, iar testele au arătat acest lucru.

Am observat faptul că acuratețea crește odată cu creșterea numărului de estimatori, în timp ce pentru RFC, mai eficient este ngram_range = (1,6).

N_estimators	**(1,6)**	**(1,7)**	**(1,8)**	**(3,7)**
100	0.732	0.724	0.7276	0.7314

N_estimators	**(1,7)**
5	0.6338
100	0.724
200	0.7314
500	0.735
1000	0.7368

Având în considerare faptul că diferențele dintre acuratețea modelului cu n_estimators =200 și cel cu 1000 este mică, am decis să folosesc soluții cu n_estimators = 200 pentru a fi sigur că modelul nu va face overfitting.

Matricea de confuzie pentru n_estimators = 200 și ngram_range = (1,6), model ce a returnat acuratețea 0.7386 , este:

```
[[1808  483  511]
 [ 123  916   20]
 [  69  101  969]]
```

3.4 Logistic Regression / Logistic Regression CV

Căutând alte modalități de predicție, am dat de un articol [1] ce menționa utilitatea Logistic Regression în cadrul Bag of Words.

Testele au arătat faptul că LR poate fi o alternativă foarte bună la RFC, acest clasificator obținând o acurateță de 0.74 - 0.75, diferența fiind făcută de solver și numărul de iterații.

Solver\max_iter	10	50	100	1000
saga	0.7426	0.7468	0.7464	0.7448
sag	0.744	0.747	0.7452	0.7446
lbfgs	0.7086	0.747	0.745	0.743
liblinear	0.7422	0.7428	0.7428	0.7428
Newton-cg	0.744	0.743	0.743	0.743

În cadrul documentației, am observat existența modelului Logistic Regression CV (Cross-Validation), un model care se antrenează și pe părți ale datelor de test, nu doar pe întreg setul de date. Folosindu-l, am observat o îmbunătățire a acurateții la un număr de 10 iterații, așa cum se observă în tabelul de mai jos.

Solver	LR	LRCV
saga	0.7426	0.748
sag	0.744	0.7454
lbfgs	0.7086	0.741
liblinear	0.7422	0.746

Așadar, am folosit LR și LRCV pe modelul ”Saga”, pe un număr de 100 de iterații, pentru a evita overfitting-ul.

Matricele de confuzie sunt:

LR

```
[[1549  345  282]
 [ 252 1035   89]
 [ 199  120 1129]]
```

LRCV

```
[[1551  317  285]
 [ 259 1063   89]
 [ 190  120 1126]]
```

4. Concluzii

Deși Language Classification este o problemă clasică de ML, încă nu există o modalitate perfectă de a o rezolva, dar, cu ajutorul antrenării modelelor pe mai multe date și modificarea indicilor, se poate scoate un procent destul de bun pentru a fi de folos în cazuri reale.

5. Bibliografie

[1] Kumar, A., *"Python – Text Classification using Bag-Of-Words Model"* https://vitalflux.com/text-classification-bag-of-words-model-python-sklearn/ , August 4, 2021

[2] scikit-learn Documentation, https://scikit-learn.org/stable/index.html