Ifrosă Diana-Maria, grupa 251

Documentatie competitie Kaggle

Tema competitiei:

Clasificarea textelor in 3 clase (identificatori 1,2,3). Textele pe care are loc procesul de invatare nu fac parte din nicio limba cunoscuta omului, ci este una "extraterestra".

Procesarea si citirea datelor

Datele au fost impartite in date de train (15 mii) si date de validare (5 mii).

Citirea datelor de train si test a fost facuta din fisiere in care in dreptul fiecarui text/eticheta este atribuit un ID. Pe baza acestor ID-uri am generat fisierele cu predictii, pentru a identifica exact textul vizat. Citirea a fost facuta linie cu linie, din fisierele deschise sub codificarea 'utf8'. Pentru fiecare linie de train (pentru sample-uri si label-uri) am despartit linia in 2 parti, pentru a elimina ID-ul, pastrand doar textul si eticheta corespunzatoare. Atat textele cat si etichetele au fost salvate in structuri de tipul np.array(). In cazul fisierului de test (cu sample-uri) am pastrat ID-urile intr-o lista pentru a le putea afisa dupa ce au fost facute predictiile. In plus, in toate cazurile am eliminat caracterele de tip spatiu (functia strip) de la inceputul si sfarsitul liniilor.

Fisierul final cu predictii este de tip CSV, fiecare linie fiind identificata prin ID, dupa care este precizat label-ul corespunzator textului.

Singura preprocesare a datelor a fost transformarea tuturor literelor in litere mici (parametru de tip lowercase in cadrul aplicarii modelelor). In linii mari, s-a observat o diferenta minima de performanta, astfel incat datele fara aceasta preprocesare au generat o acuratete mai ridicata.

Reprezentarea datelor

Fiind vorba despre procesare de text, am folosit reprezentari de tipul Bag of Words, TF-IDF sau N-grame.

Bag of Words este o reprezentare ce consta in distingerea cuvintelor din cadrul textului (fie manual, prin a delimita in functii de spatii, fie cu clase specializate precum *CountVectorizer* din *sklearn*) si aflarea frecventei acestora in intreg documentul (practic, feature-urile sunt date de aceste frecvente). In final, se obtine un vector de dimensiune egala cu lungimea vocabularului ce cuprinde toate cuvintele.

TF-IDF este o reprezentare similara Bag of Words, insa ia in considerare si cat de frecvente sunt cuvintele, in sensul in care sunt atribuite ponderi mai mari sau mai mici in functie de cat de intalnit este acel cuvant in dictionar. In acest sens, am folosit clasa *TfidfVectorizer* din *sklearn.*

N-grame este o reprezentare ce consta in luarea unor secvente continue de n caractere (sau cuvinte intregi) si calcularea similaritatii intre clase. In general, am ales sa testez modele cu n in intervalul [3,10] in cazul literelor (adica fie sa calculez similaritatea intre n-grame unde n este fixat, fie intre multimi de n-grame- de exemplu unigrame si bigrame etc.), si [2,5] in cazul cuvintelor intregi (la fel cu n fixat sau pentru interval de la 1 la n). Acest lucru a fost posibil datorita parametrului *ngram_range* din clasele precizate anterior. In cazul caracterelor, secventele au fost completate cu spatii atunci cand lungimea lor nu era suficienta. Parametrul utilizat pentru a implementa aceasta reprezentare este *analyzer*, setat fie pe *word*, fie pe *char*, din cadrul clasei *CountVectorizer*.

Modele folosite

Acestea sunt:

1. SVM (Support vector machines) cu ajutorul clasei *SVC* din *sklearn.*
2. Naïve Bayes cu ajutorul clasei *MultinomialNB* din *sklearn* (distributie multinomiala)
3. Multi-layer perceptron cu ajutorul clasei *MLPClassifier* din *sklearn*

1. SVM

Abordare 1 (submisie 1 pe Kaggle):

Reprezentarea folosita a fost Bag of Words, implementata manual, prin a construi un vocabular cu toate cuvintele intalnite in fisierul vizat, iar mai apoi a atribui fiecaruia numarul de aparitii pe fiecare linie (sample). Astfel, functia implementata returneaza o matrice de valori intregi (un np.array()) pe care am folosit-o la antrenare, respectiv predictii. Datele nu au fost preprocesate.

Am ales pentru acest model functia de decizie one-versus-one pentru a obtine o mai buna acuratete, intrucat am lucrat pe clasificare cu 3 clase (multi-class). Parametrii modelului au fost, in afara acestei modificari, cei default (precum *kernel:'rbf', regularizare: 1.0, random_state:None* etc.).

Punctajul obtinut pe Kaggle in timpul competitiei: **0.63935**, la finalul competitiei: **0.62876.**

Matrice de confuzie pentru cea mai mare acuratete:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1651	221	128
Planeta 2	611	790	99
Planeta 3	585	111	804

Abordare 2:

Reprezentarea folosita a fost n-grame pe caractere cu n in intervalul [4,6], folosind *CountVectorizer(analyzer = 'char', lowercase = False, ngram_range(n,n)).*

Nu a fost facuta nicio preprocesare a datelor.

Cea mai buna acuratete a fost obtinuta pentru **n = 4.**

Cea mai scazuta acuratete a fost obtinuta pentru n = 6.

Matrice de confuzie pentru cea mai mare acuratete:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1692	212	96
Planeta 2	599	798	103
Planeta 3	454	74	972

Comparatie intre cele 2 abordari:

Acuratete (obtinuta pe validare**)**	**Abordare 1**	**Abordare 2**		
		N = 4	*N = 5*	*N = 6*
	0.649	0.6924	0.6914	0.6872

2. ***Naïve Bayes***

Observatie pentru toate abordarile:

Modelul folosit a fost cel al distributiei multinomiale: *MultinomialNB()*, avand parametrii default precum *alpha:1.0, fit_prior: True.*

Abordarea 1 (submisie 2 pe Kaggle):

Reprezentare: Bag of Words, implementata manual cu date nepreprocesate.

Matrice de confuzie pentru cea mai mare acuratete:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1822	98	80
Planeta 2	161	1277	62
Planeta 3	170	27	1303

Abordarea 2 (submisiile 3 si 4 pe Kaggle):

Reprezentare: Bag of Words cu *CountVectorizer*.

Am comparat acuratetea intre datele neprocesate (submisie 4) si cele procesate utilizand lowercase (submisie 3).

De asemenea, am ales sa lucrez cu un procent din cele mai frecvente cuvinte (*max_features*), procent cuprins intre 10% si 90% (valori din 10% in 10%), pentru a afla numarul optim de features.

Cea mai buna acuratete a fost obtinuta pentru **procentul de 40%, fara lowercase.**

Cea mai scazuta acuratete a fost obtinuta pentru procentul de 10%, cu lowercase.

Abordarea 3 (submisie 5 pe Kaggle):

Reprezentare: n-grame pe caractere cu n in intervalul [3,20].

Am comparat acuratetea intre datele neprocesate si cele procesate utilizand lowercase.

Am folosit clasa CountVectorizer, pastrand majoritatea parametrilor default:

CountVectorizer(analyzer='char', lowercase=preprocessing, ngram_range=(n,n))

Cea mai buna acuratete a fost obtinuta pentru **n = 5, fara lowercase.**

Cea mai scazuta acuratete a fost obtinuta pentru n=20, fara lowercase.

Matrice de confuzie pentru cea mai mare acuratete:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1586	279	135
Planeta 2	360	1031	109
Planeta 3	323	87	1090

Abordarea 4 (submisie 10 pe Kaggle):

Reprezentare: n-grame cu TF-IDF pe cuvinte; am ales n in intervalul [2,7] si am considerat multimi de n-grame.

Am comparat acuratetea intre datele neprocesate si cele procesate utilizand lowercase.

Am folosit clasa TfidfVectorizer, pastrand majoritatea parametrilor default:

TfidfVectorizer(analyzer='word', lowercase=preprocessing, ngram_range=(1, n))

Cea mai buna acuratete a fost obtinuta pentru **n = 2, fara lowercase.**

Cea mai scazuta acuratete a fost obtinuta pentru n = 2, cu lowercase.

Abordarea 5 (submisie 8 pe Kaggle):

Reprezentare: n-grame pe cuvinte, cu n in intervalul [2,7] (multimi de n-grame).

Am comparat acuratetea intre datele neprocesate si cele procesate utilizand lowercase.

Am folosit clasa CountVectorizer, pastrand majoritatea parametrilor default:

CountVectorizer(analyzer='word', lowercase=preprocessing, ngram_range=(1,n))

Cea mai buna acuratete a fost obtinuta pentru **n = 2, cu lowercase.**

Cea mai scazuta acuratete a fost obtinuta pentru n = 7, cu lowercase.

Comparatie intre abordari:

Acuratete maxima (validare)	**Abordare 1**	**Abordare 2**	**Abordare 3**	**Abordare 4**	**Abordare 5**
	0.8804	0.6894	0.7414	0.5782	0.7074
Acuratete minima(validare)	-	0.6696	0.623	0.5758	0.6792

Acuratete intermediara	**Abordare 1 (solutie selectata)**	**Abordare 2**		**Abordare 3**	**Abordare 4**	**Abordare 5**
		Submisie 3	Submisie 4			
	0.74685	0.69773	0.70028	0.74219	0.51167	0.71874
Acuratete finala	0.73958	0.70280	0.70105	0.74235	0.52997	0.71414

*Acuratete intermediara = in timpul concursului

*Acuratete finala = dupa terminarea concursului

3. Multi-layer perceptron

Abordarea 1 (submisie 6 pe Kaggle):

Reprezentare: Bag of Words cu un un anumit procent din cele mai frecvente cuvinte, intre 10% si 90%, cu *CountVectorizer.*

Am comparat acuratetea intre datele neprocesate si cele procesate utilizand lowercase.

Straturi ascunse: am testat pentru diverse configuratii pentru a obtine optimul: (32,), (32, 64), (32, 64, 128), (64, 64, 64).

Rata de invatare: am pornit de la cea default 0.0001, si pe langa aceasta am testat pentru variatiuni mici, precum 0.01 si 0.001.

De asemenea, am folosit optiunea de **early stopping** pentru a gasi epoca ideala la care sa se opreasca procesul. Am salvat aceasta epoca de la modelul cu cea mai buna acuratete obtinuta, iar in final am antrenat modelul doar pana la acea epoca (early_stopping = False).

Solver: cel default, "**adam**". Batch size: cel default: min(200, n_samples).

Cea mai buna acuratete a fost obtinuta pentru **procentul 70%, fara lowercase, rata de invatare 0.0001, straturile ascunse (32,64), epoca 22.**

Cea mai scazuta acuratete a fost obtinuta pentru procentul 30%, cu lowercase, rata de invatare 0.01, straturile ascunse (32,64,128), epoca 17.

Abordarea 2 (submisie 12 pe Kaggle):

Reprezentare: Bag of Words cu un anumit procent din cele mai frecvente cuvinte, intre 30% si 60%, cu *CountVectorizer*. Am comparat acuratetea intre datele neprocesate si cele procesate utilizand lowercase.

Solver: "sgd" (**stochastic gradient descent**), iar pentru rata de invatare am setat parametrul "adaptive", pentru a se micsora de fiecare data cand la 2 epoci consecutive fie nu creste scorul pentru validare, fie nu scade "loss"-ul pentru partea de antrenare.

Batch size: cel default, min(200, n_samples).

Straturi ascunse: am testat pentru diverse configuratii pentru a obtine optimul: (32,), (32, 64), (32,64,128).

Rata de invatare: am pornit de la cea default 0.0001, si pe langa aceasta am testat si pentru rate mai mari, precum 0.01 sau 0.1.

La fel ca la abordarea precedenta, am setat *early_stopping* True si am retinut epoca la care se opreste.

Cea mai buna acuratete a fost obtinuta pentru **procentul 60%, cu lowercase, rata de invatare 0.01, straturile ascunse (32,64), epoca 32.**

Abordarea 3 (restul submisiilor de pe Kaggle):

In aceasta abordare, m-am axat pe gasirea parametrilor optimi pentru n-grame pe caractere. Am incercat sa gasesc, folosind *CountVectorizer*, n-ul optim cautand in intervalul [3,10], atat pentru n-grame simple, cat si pentru multimi de n-grame.

Datele au fost fie neprocesate, fie procesate folosind lowercase. In cele mai multe cazuri, optiunea de a nu aplica lowercase s-a dovedit mai buna.

Solver-ul folosit pentru toate incercarile a fost cev default, "adam".

Straturi ascunse: am testat pentru diverse configuratii pentru a obtine optimul: (32,), (32, 64), (32,64,128), parametrul default: (100,), (32, 64, 128,256), (64,128), (64,128,256), ultimele 3 fiind testate in momentul in care am modificat si *batch size* in 128 sau 64, pana atunci fiind cel default anume min(200, n_samples).

Early_stopping a fost setat peTrue si am folosit epoca la care se opreste cel mai bun model pentru a antrena pe datele de test.

Am trimis ca submisii predictiile date de cele mai bune modele, dar si unele cu potential, dar care nu dadeau acuratete maxima.

Exemple de cele mai bune modele din cadrul acestei abordari :

Numar submisie	**Straturi ascunse**	**Rata de invatare**	**Epoca**	**N-ul**	**Preprocesare**	**Batch size**	**Acuratete (pe datele de validare)**
7	(32,)	0.0001	12	5	-	default	0.7422
11	(32, 64, 128)	0.0001	18	5	-	default	0.7402
18	(32, 64, 128)	0.0001	35	6	-	128	0.7396
16	Default = (100,)	0.0001	Am folosit early_stopping la antrenare pentru test	6	-	default	0.747
17	(64, 128)	0.00001	42	5	-	128	0.7454
15	(32,)	0.0001	16	5	-	64	0.7452
14	(32,64,128)	0.00001	46	5	-	128	0.7508

Compararea acuratetii cu datele primite pe site:

Numar submisie	**Acuratete intermediara**	**Acuratete finala**
7	0.76437	0.74881
11	0.75801	0.74635
14	0.75933	0.74390
15	0.76083	0.74622
16	0.75568	0.75215
17 **(solutie selectata)**	0.76833	0.74733
18	0.75767	0.74383

Ifrosă Diana-Maria, grupa 251

Matrice de confuzie pentru submisia 7:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1564	255	181
Planeta 2	379	992	129
Planeta 3	302	79	1119

Matrice de confuzie pentru submisia 15:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1531	305	164
Planeta 2	343	1034	123
Planeta 3	292	83	1125

Matrice de confuzie pentru submisia 17:

Predictii => **Etichete reale ↓**	**Planeta 1**	**Planeta 2**	**Planeta 3**
Planeta 1	1577	282	141
Planeta 2	348	1023	129
Planeta 3	292	81	1127

Antrenarea si alegerea celui mai bun model

Antrenarea a fost facuta pe datele de train, iar mai apoi am generat predictii pentru datele de validare, pentru a alege cel mai bun model, dat de cea mai buna acuratete obtinuta. Acuratetea am masurat-o folosind *np.mean(predictions == validation_labels).* In final, dupa alegerea celui mai bun model, in majoritatea submisiilor, am atrenat pe datele de train & cele de validare (am concatenat cei 2 np.arrays) pentru a genera predictii pe datele de test.