**Documentatie concurs Kaggle**

**Neagu Matei**

**Grupa 232**

1. **Citirea datelor**

Fiecare fisier .txt de intrare contine, mai intai, un id al obiectului, apoi un caracter „tab”, apoi fie o eticheta, fie o fraza intr-o limba extraterestra. Asadar, pentru citire am utilizat obiecte de tip fisier obisnuit din Python. Prin ele, am extras toate liniile din fisierele de intrare si le-am parcurs, separand, pentru fiecare linie, continutul dupa caracterul „tab” si salvand cele 2 parti ale liniei in liste diferite, care in final au fost convertite in „ndarrays” pentru utilizarea lor in modelele de clasificare.

1. **Preprocesarea datelor & reprezentarea atributelor**
2. **Descriere**

Intrucat obiectele de clasificare iau forma unor string-uri cu diverse caractere, am optat pentru utilizarea algoritmului „bag of words”. Prin bag of words, sunt parcurse toate cuvintele (siruri de caractere separate prin spatii) din fisierul de antrenare pentru a putea memora o lista care contine toate cuvintele distincte gasite si un dictionar in care o cheie este un cuvant gasit, iar valoarea este pozitia cuvantului in lista mentionata anterior. In acest fel, se pot reprezenta obiectele astfel: un obiect va fi transformat intr-un vector de frecventa de lungimea listei cu cuvinte distincte, in care pe pozitia i se va memora numarul de aparitii in fraza curenta a cuvantului de pe pozitia i din lista cu cuvinte.

O alta metoda de reprezentare folosita este cea a transformarii frazelor folosind CountVectorizer, o implementare bazata pe „bag of words” din biblioteca sklearn. In plus fata de implementarea proprie descrisa mai sus, aceasta clasa a permis o personalizare mai rapida a atributelor obiectelor.

1. **Parametri CountVectorizer**

Constructorul clasei CountVectorizer admite urmatorii parametri, care au ajutat in modelarea reprezentarii:

* **lowercase** -> specifica daca toate literele vor fi convertite in minuscule
* **strip\_accents** -> scoate accentul pentru caracterele care au un corespondent direct in ASCII, pentru orice caracter, sau pentru niciunul
* **binary** -> in cazul in care este setat la valoarea True, un atribut nu va mai fi frecventa cuvantului, ci 1 daca acesta apare in fraza sau 0 in caz contrar
* **max\_features** -> se poate specifica un numar intreg care sa devina numarul maxim de atribute pe care le poate avea un obiect (nespecificat, numarul maxim este egal cu numarul de cuvinte distincte din toate obiectele)
* **max\_df** -> este un numar real intre 0 si 1. Daca un cuvant are frecventa (procentajul) mai mare decat aceasta valoare, atunci nu mai ia parte la reprezentari.
* **min\_df** -> la fel ca la max\_df, cuvintele cu frecventa mai mica decat aceasta valoare nu sunt considerate
* **analyzer** -> daca are valoarea „word”, atunci sirurile de caractere preluate pentru care se calculeaza frecventa sunt cuvinte. Daca are valoarea „char”, atunci sunt preluate caractere individuale.
* **ngram\_range** -> un tuplu de forma (x, y). Frecventa este calculata pentru secventele de lungime minim x si maxim y cu cuvinte/caractere consecutive din obiecte.

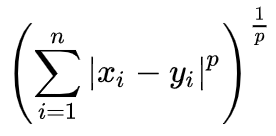
1. **Algoritmul KNN**
2. **Descriere model**

Algoritmul celor mai apropiati K vecini este folosit pentru invatarea automata supervizata, in special pentru sarcinile de clasificare. Algoritmul stocheaza, dintr-o colectie de date de antrenare, fiecare obiect deja clasificat, urmand ca unui obiect nou sa i se atribuie clasa cea mai frecvent intalnita printre cei mai apropiati K vecini ai sai, acestia fiind determinati printr-o metoda de calcul specificata.

1. **Parametri model**

Pentru a utiliza acest clasificator in solutie, am folosit implementarea sa din pachetul sklearn.neighbors: KNeighborsClassifier. Constructorul clasei admite o serie de parametri, dupa cum urmeaza:

* **n\_neighbors** -> reprezinta valoarea intreaga K, care determina cati primi vecini sa fie luati in considerare atunci cand se gaseste clasa unui obiect nou. In general, mai multi factori contribuie in setarea acestui parametru: o valoare prea mare ar putea duce atat la un timp de executare ridicat, cat si la fenomenul de „underfitting”; o valoare foarte mica va duce, de cele mai multe ori, la „overfitting”. Totodata, sunt importante si paritatea, si raportul dintre K si numarul de clase. Daca K este par, creste probabilitatea de a avea 2 clase cu aceeasi frecventa (K / 2) printre primii K vecini, iar decizia de alocare a unei clase sa fie luata la intamplare. Daca K este un multiplu al numarului de clase, atunci apare acelasi risc, de a avea toate clasele prezente cu aceeasi frecventa.
* **weights** -> defineste cat de mult cantareste apartenenta individuala la o clasa a unuia dintre cei K vecini. Parametrul poate fi setat ca „uniform”, astfel ca fiecare apartenenta cantareste in mod egal, sau ca „distance”, care face ca apartenenta vecinilor mai apropiati sa fie mai importanta.
* **metric** -> exprima ce formula este utilizata in calcularea distantei dintre 2 obiecte
* **p** -> in cazul optarii pentru formula generalizata „Minkowski” drept metrica, p va fi valoarea numarului inlocuit in formula respectiva.

 -> formula Minkowski. Pentru p=1, formula devine cea a distantei Manhattan, iar pentru p=2, avem distanta euclidiana.

* **algorithm** -> specifica algoritmul folosit pentru a gasi vecinii cei mai apropiati. Anumiti algoritmi pot scadea semnificativ complexitatea timpului („brute force” este mai incet decat „kd tree” sau „ball tree”), insa nu sunt neaparat si potriviti pentru sarcina de lucru, astfel ca setarea parametrului la „auto” duce la o determinare automata a celui mai bun algoritm.

1. **Hiperparametrizare**

Hiperparametrizarea pentru modelul KNN a inceput prin gasirea valorilor k pentru care acuratetea obtinuta pe datele de validare era cea mai inalta. Asadar, am testat valorile 5, 4 si 7 pentru k, tinand astfel cont de bunele practici legate de acest algoritm (evitand cazurile in care este mai probabil sa avem 2 clase cu aceeasi frecventa).

In urma obtinerii unor rezultate mai bune pentru valorile mai mici (5 si 4), am hotarat testarea si pentru valorile 3, 2, acestea din urma dovedindu-se chiar mai bune. Ulterior, am sesizat o imbunatatire prin a configura modelul pentru a tine cont de distanta vecinilor atunci cand se hotaraste clasa unui obiect. In mod implicit, algoritmul foloseste formula Minkowski cu p = 2. Schimbarea in distanta Manhattan pentru cele mai bune configuratii (k = 3 si k = 2) a dus la performante mai bune.

1. **Rapoarte de acuratete si matrice de confuzie**

|  |  |  |  |
| --- | --- | --- | --- |
| **n\_neighbors** | **weights** | **p** | **acuratete** |
| 7 | uniform | 2 | 0.4034 |
| 5 | uniform | 2 | 0.4302 |
| 4 | uniform | 2 | 0.4494 |
| 3 | uniform | 2 | 0.4576 |
| 2 | uniform | 2 | 0.4594 |
| 4 | distance | 2 | 0.4668 |
| 3 | distance | 2 | 0.4716 |
| 2 | distance | 2 | 0.4704 |
| 3 | distance | 1 | 0.4742 |
| 2 | distance | 1 | 0.4704 |

Cea mai buna configuratie s-a dovedit a fi cea gasita pe penultima linie a tabelului. Pentru aceasta, avem matricea de confuzie:

|  |  |  |  |
| --- | --- | --- | --- |
| Prezis -> | 1 | 2 | 3 |
| 1 | 1459 | 525 | 16 |
| 2 | 730 | 754 | 16 |
| 3 | 1092 | 250 | 158 |

Clasele 2 si 3 par a fi confundate cu clasa 1 foarte des. Incercam si ultima configuratie

|  |  |  |  |
| --- | --- | --- | --- |
| Prezis -> | 1 | 2 | 3 |
| 1 | 1474 | 451 | 75 |
| 2 | 744 | 679 | 77 |
| 3 | 1085 | 216 | 199 |

1. **Multinomial Naive Bayes**
2. **Descriere model**

Modelul, bazat pe teorema lui Bayes, este utilizat cu precadere in sarcinile de clasificare, avand rezultate bune in special in cadrul procesarii limbajului natural. Avand la dispozitie obiecte cu etichete pentru antrenare, se calculeaza pentru fiecare clasa cate atribute din fiecare tip distinct de atribut se regasesc. Astfel, se poate calcula probabilitatea ca un anumit atribut sa se gaseasca intr-un obiect dintr-o clasa specifica. Apoi, este stabilita probabilitatea „a priori” a claselor, adica se determina care este probabilitatea ca un obiect oarecare sa apartina unei clase.

Atunci cand un obiect nou (din datele de testare) este clasificat, se calculeaza probabilitatea „a posteriori” pentru fiecare clasa in raport cu obiectul, iar clasa cu scorul cel mai mare i se atribuie acestuia. Pentru a calcula probabilitatea „a posteriori”, se inmulteste probabilitatea „a priori” a clasei cu probabilitatea ca un atribut al obiectului sa se gaseasca in clasa respectiva, pentru fiecare atribut al sau.

1. **Parametri model**

* **alpha** -> in situatii mai complexe, in care un obiect nou care apartine unei clase contine un atribut deloc gasit in acea clasa in cadrul datelor de antrenare, va avea intreaga probabilitate „a posteriori” egala cu 0, intrucat probabilitatea de a gasi acel atribut in clasa a fost calculata ca fiind 0. Pentru a evita aceste situatii, se poate specifica parametrul alpha – un numar care este adunat la frecventa fiecarui atribut in prima faza a calculului.
* **fit\_prior** -> daca este setat ca adevarat, atunci probabilitatea „a priori” a claselor este calculata in functie de numarul de obiecte din fiecare clasa in datele de antrenare. Daca este fals, atunci sunt atribuite probabilitati uniforme.
* **class\_prior** -> se specifica manual probabilitatile „a priori” ale claselor

1. **Hiperparametrizarea modelului, corelata cu cea a CountVectorizer**

Pentru inceput, am incercat folosirea Multinomial NB cu parametri default (alfa 1, fit\_prior true, class\_prior false) si implementarea proprie a bag of words, descrisa la punctul 2. Rezultatele au fost satisfacatoare, asa ca am continuat prin a incerca sa modific cativa dintre parametri modelului. Am observat ca folosirea parametrului fit\_prior cu valoarea True duce la overfitting, astfel ca l-am schimbat in False iar rezultatele s-au imbunatatit. Schimbarea parametrului alfa ducea, rapid, la underfitting.

„Naivitatea” modelului de clasificare provine din faptul ca acesta nu tine cont de ordinea cuvintelor sau de modul lor de grupare pentru a interpreta o secventa de text, ceea ce este total diferit fata de modul in care ar actiona un om. Asadar, am folosit CountVectorizer pentru a incerca imbunatatirea acestui comportament prin schimbarea modului de reprezentare al obiectelor.

Initial, am incercat folosirea CountVectorizer cu parametrul max\_features setat la diverse valori. Valorile intre 16-20k au oferit o acuratete sub cea a implementarii proprii. Parametrul binary setat la True a imbunatatit semnificativ performanta modelului, alaturi de folosirea n-gramelor cu lungimi intre 1 si 2 si convertirea literelor in minuscule. Cu toate acestea, cea mai mare schimbare a fost adusa de scoaterea completa a parametrului max\_features.

O alta schimbare importanta a fost adusa de modificarea tipului ngramelor, din cuvinte in caractere. Pentru acestea, intervalul de lungime al ngramelor a trebuit marit (intrucat una sau 2 litere consecutive nu sunt concluzive), iar diferenta dintre capete a trebuit pastrata mai mare, pentru a evita overfitting-ul.

In urma acestor incercari, o imbunatatire a fost adusa si prin antrenarea modelului pe datele de validare. Rezultatele in cadrul competitiei au devenit mai bune.

Constatand ca modificarea parametrilor CountVectorizer nu mai aduce imbunatatiri, am hotarat, in final, sa folosesc GridSearchCV pentru a gasi cei mai buni parametri ai Multinomial NB. Am sugerat cele mai bune valori constatate in timp pentru alpha si valorile True si False pentru fit\_prior.

In final, configuratia alpha = 0.1 si fit\_prior=True a fost constatata antrenand doar pe datele de antrenare, iar configuratia alpha = 0.1 si fit\_prior=False a fost constatata antrenand si pe datele de validare. In mod clar, fit\_prior=True ducea la overfitting, astfel ca cele mai bune rezultate pe datele de testare au fost obtinute folosind fit\_prior = False.

1. **Rapoarte de acuratete si matricea de confuzie**

Matricea de confuzie pentru ultima linie a tabelului:

|  |  |  |  |
| --- | --- | --- | --- |
| **Prezis ->** | **1** | **2** | **3** |
| **1** | **1557** | **276** | **167** |
| **2** | **293** | **1086** | **121** |
| **3** | **250** | **80** | **1170** |

Matricea de confuzie daca schimbam fit\_prior in False (fara a fi antrenat pe datele de validare)

|  |  |  |  |
| --- | --- | --- | --- |
| **Prezis ->** | **1** | **2** | **3** |
| **1** | **1562** | **275** | **163** |
| **2** | **295** | **1084** | **121** |
| **3** | **251** | **79** | **1170** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **alpha** | **fit\_prior** | **reprezentare** | **lower** | **bin** | **ngram\_range** | **max\_feat** | **analyzer** | **strip** | **acc** |
| **1** | **False** | **proprie** |  |  |  |  |  |  | **0.732** |
| **0.5** | **False** | **proprie** |  |  |  |  |  |  | **0.7282** |
| **1** | **True** | **count** | **T** | **T** | **(1, 1)** | **16k** | **word** |  | **0.6908** |
| **1** | **True** | **count** | **T** | **T** | **(1, 2)** | **20k** | **word** |  | **0.6958** |
| **1** | **True** | **count** | **F** | **T** | **(3, 6)** |  | **char** | **unicode** | **0.7478** |
| **0.1** | **True** | **count** | **F** | **T** | **(3, 6)** |  | **char** | **None** | **0.7632** |