Alexandru Emil-Adrian, grupa 354

Documentatie proiect Inteligenta Artificiala
Translation Source Language Identification

Retele neuronale

MLPClassifier

Pentru a clasifica elementele am folosit modelul multi-layer perceptron (MLP), un clasificator feedforward, care face operatiile doar de la stanga la dreapta in lista de straturi. Acesta ajuta la implementarea unei retele neuronale pentru a putea face clasificarea datelor dorite. Se importa din libraria sklearn.neural_network.

Algoritmul incepe prin selectarea unui numar de cuvinte ales, 6000, numar care reprezinta primele cele mai frecvente cuvinte din totalul de cuvinte obtinut dupa ce am calculat pentru fiecare cuvant de cate ori apare in toate fisierele si le-am sortat in ordine descrescatoare.

Am folosit ‘re.sub(‘[-.,::!?\”\’\/()_*=`]’,’’, text)’ pentru a elimina toate semnele de punctuatie din texte.

Am folosit normalizarea: v = (v - np.mean(v)) / np.std(v).

Apoi a trebuit facuta impartirea datelor pe date de antrenare si test. Am folosit 2700 de exemple pentru antrenare, 150 de exemple pentru validare si 43 de exemple pentru testare.

Ca paramentrii pentru clasificatorul MLP am folosit:

- **Hidden_layer_sizes**: Acest parametru ne ajuta sa setam numarul de straturi si de noduri pe care sa-l aiba reteaua noastra neuronala. Astfel i-am dat valorile (300, 200), ceea ce inseamna ca avem 2 straturi si 300 de noduri pe un strat, respectiv 200 pe celalalt.
- **Activation**: pentru aceasta functie de activare i-am dat functia hyperbolica TANH unde f(x)=tanh(x)
- **Solver**: i-am dat valoarea ADAM, care este valoarea by default, deoarece se comporta foarte bine pe date de antrenare relative mari(cum este si cazul de fata cu peste 2000 de exemple de antrenare)
- **Alpha** : a primit valoarea 0.001
- **Max_iter**: numarul de epoci a luat valoarea 200, valoare by default a clasificatorului

Dupa setarea parametriilor am antrenat pe datele de antrenare, testandu-le pe datele de validare, dupa care am concatenat datele de antrenare si cele de validare, am facut antrenarea pe acestea si am testat pe datele de testare.

Timpul de executie al antrenarii a fost de 36 de secunde.

Performanta obtinuta pe datele publice de pe Kaggle a fost de 91,137%.

Matricea de confuzie asociata facuta pe datele de test :

[11 0 0 0 1 0 0 0 0 0 0]
[0 14 0 0 0 0 0 0 0 0 0]
[0 0 17 0 0 0 0 0 0 0 0]
[0 0 0 14 0 0 0 0 0 0 0]
[0 0 0 0 13 0 0 0 0 0 0]
[1 0 0 0 0 12 0 0 0 0 0]
[0 0 0 0 0 0 8 0 0 0 0]
[0 0 0 0 0 0 0 7 0 0 0]
[0 0 0 0 0 0 0 0 9 1 0]
[0 0 0 0 0 1 0 0 0 11 0]
[0 0 0 0 0 0 0 0 0 0 13]

K-Fold Cross Validation este folosit pentru a imparti setul de date pe care il ai la dispozitie in k sectiuni. Fiecare sectiune este apoi folosita ca set de test. In cazul nostru K = 10, la prima iteratie, prima sectiune este folosita pentru test iar restul pentru antrenare, la a doua iteratie, a doua sectiune este folosita pentru testare iar restul sectiunilor pentru testare. Facem asta pana cand toate sectiunile au fost folosite ca test iar la final facem o medie a scorurilor obtinute pentru fiecare din cele 10 si o afisam.

Scorul obtinut de 10 fold cross-validation este de 0.8864340044742729

SVM

Am ales clasificatorul SVM din libraria sklearn. Am folosit Kernel linear deoarece este folosit atunci cand avem date linear separabile. Avantajul folosirii acestuia este ca este mai rapid decat celelalte kernel.

Hyperparametrul C este folosit pentru a determina cat de mare, sau cat de mica sa fie distanta intre punctele linear separate si dreapta ce le separa. Pentru o valoare mare data lui C, marginea va fi mai mica, iar pentru o valoare mica, marginea va fi una mare. In aplicarea modelului pe datele de pe Kaggle am folosit C=10.

Algoritmul incepe prin selectarea unui numar de cuvinte ales, 6500, numar care reprezinta primele cele mai frecvente cuvinte din totalul de cuvinte obtinut dupa ce am calculat pentru fiecare cuvant de cate ori apare in toate fisierele si le-am sortat in ordine descrescatoare.

Am folosit normalizarea: v = (v - np.mean(v)) / np.std(v)

Numarul de exemple pentru antrenare este de 2800 iar cel pentru validare este de 60, pentru test ramanand restul de exemple din total.

Mai intai antrenam SVM cu SVC pe datele de antrenare pentru fiecare valoare a lui C din intervalul {0.01, 0.1, 1, 10, 100} dupa care le testam pe datele de validare pentru a observa acuratetea.

Dupa aceea repetam acelasi procedeu doar ca datele de antrenare de data aceasta vor consta in concatenarea datelor de antrenare folosite anterior cu cele de validare, iar testarea se va face pe diferenta ramasa din totalul de exemple.

Folosim un timer pentru a vedea timpul de executie al antrenarii, acesta consta in 2 parametrii, unul care retine timpul la momentul inceperii executiei iar celalalt la momentul terminarii executiei, astfel putem afisa diferenta dintre cele doua momente ca fiind timpul de executie al antrenarii. In urma executiei acesta a fost de 95 de secunde.

Pentru setul de date public de pe Kaggle am obtinut folosind acest clasificator o acuratete de 81,438%.