Documentatie Proiect - Inteligenta Artificiala

Petrescu Cosmin, Grupa 243

April 6, 2021

Cerinta

Antrenarea unui model pentru clasificarea imaginilor monocrome in 9 clase. Setul de date de antrenare contine 30,000 de imagini, cel de validare 5,000 de imagini, iar cel de testare inca 5,000 de imagini.

Observatie

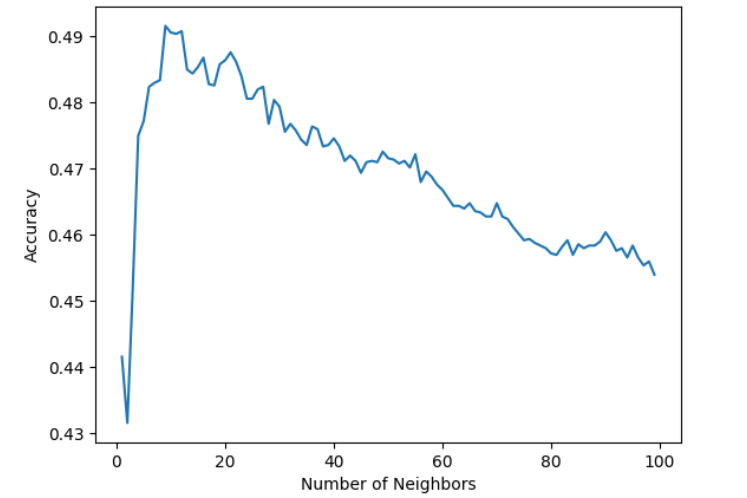
Progresul acestui proiect mentine o paralela cu laboratorul, alegand astfel sa utilizez modelele abordate in cadrul acestuia. De asemenea, o parte din notiunile ce urmeaza a fi prezentate sunt preluate din materialele de laborator. Pentru utilizarea modelelor prezentate am folosit biblioteca *scikit-learn* si platforma *TensorFlow*.

1. Naive Bayes

Clasificatorul Naive Bayes este un model ce se bazeaza pe teorema lui Bayes pentru determinarea probabilitatii apartenentei unui exemplu la o anumita clasa. In cazul imaginilor, acest clasificator presupune faptul ca pixelii reprezinta atribute **independente**. In urma antrenarii s-a observat ca cele mai bune scoruri pe setul de validare au fost de: 0.3904 pentru Multinomial Naive Bayes si 0.3992 pentru Gaussian Naive Bayes.

2. K-Nearest Neighbors

Al doilea tip de clasificator testat este K-Nearest Neighbors. Acesta calculeaza distantele dintre imaginea data si celelalte imagini ale caror clase sunt cunoscute. Sunt selectate apoi cele mai apropiate K imagini si se determina clasa predominanta. Acest model a fost testat cu distanta euclidiana si $K \in \{1, 2 \dots 100\}$, obtinand un scor pe setul de validare de 0.4916, atins cu parametrul K=9. Evolutia scorului in functie de parametrul K este evidentiata in graficul urmator:



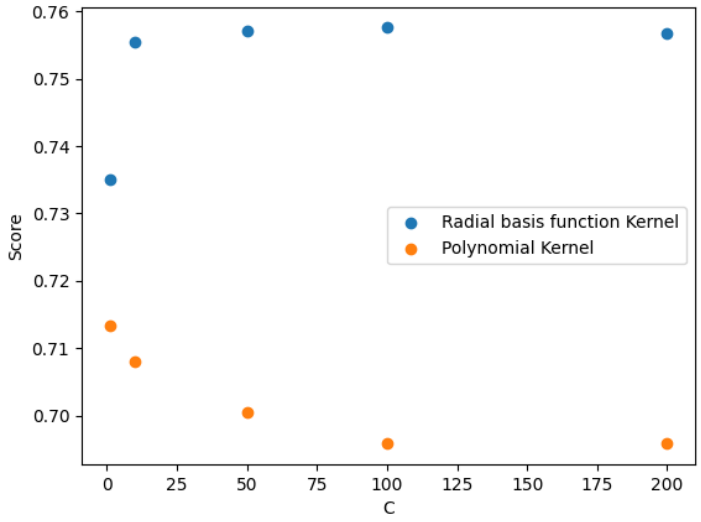
3. Support Vector Machines

Support Vector Machine (SVM) este un model care are ca rol gasirea unui hiperplan de margine maxima pentru a separa elementele a doua clase. Pentru clasificarea multipla exista doua abordari: *ONE vs ALL* sau *ONE vs ONE*. Implementarea din biblioteca scikit-learn foloseste abordarea *ONE vs ONE*, deci se vor antrena $\frac{nr\_clase \cdot (nr\_clase-1)}{2}$. In cazul in care datele nu sunt liniar separabile, se folosesc functiile kernel. Acestea functioneaza astfel:

- Datele sunt scufundate într-un spațiu (Hilbert) cu mai multe dimensiuni.
- Relațiile liniare sunt căutate în acest spațiu.

Pe langa functia kernel, un alt parametru relevant este parametrul de penalitate pentru eroare C. Cand C este mare, se va alege un hiperplan cu o margine mai mica, daca acesta are rezultate mai bune pe setul de antrenare iar cand C este mic se va alege un hiperplan cu o margine mai mare, chiar daca acesta duce la clasificarea gresita a unor puncte din setul de antrenare.

Mai jos se poate vizualiza evolutia scorului pe setul de validare in functie de parametrul C si functia kernel:



4. Multilayer Perceptron

Multilayer Perceptron (MLP) reprezinta o retea de perceptroni multistrat, de tip *feedforward*. Aceasta contine un strat de perceptroni numit strat de intrare, fiind singurul strat care primeste date de intrare din exterior, un strat de iesire si cel putin un strat ascuns. Aceasta foloseste un algoritm de antrenare numit *backpropagation* bazat pe *gradient descent*, calculand gradientul functiei de pierdere in raport cu ponderile retelei.

Pentru preprocesarea imaginilor am utilizat *StandardScaler* pentru standardizarea atributelor (pixelilor), acesta scazand media si impartind la deviatia standard. Pentru alegerea hiperparametrilor, am folosit metoda *Random Search* care, plecand de la un dictionar cu elemente de tip *hiperparametru : [valoare1, valoare2 ...]* gaseste o combinatie a hiperparametrilor care ofera cel mai bun scor pentru un anumit model. Acest lucru se realizeaza in principal prin generarea aleatoare a unor combinari si evaluarea modelului be configuratia generata.

Dictionarul hiperparametrilor pentru Random Search este:

- hidden_layer_sizes: [(1024, 9), (1024, 512, 9), (1024, 690, 690, 9)]
- activation: ['tanh', 'relu', 'logistic']
- solver: ['sgd', 'adam']
- learning_rate_init: [0.0001, 0.001, 0.01, 0.1]
- alpha: [0.0001, 0.05]
- learning_rate: ['constant', 'adaptive']
- momentum:[0.9, 0.8]

Dupa rularea algoritmului Random Search, am obtinut valorile optime urmatoare, cu un scor pe setul de validare de **0.7746**:

- hidden_layer_sizes: (1024, 690, 690, 9)
- activation: 'relu'
- solver: 'adam'
- learning_rate_init: 0.0001
- alpha: 0.05
- learning_rate: 'constant'
- momentum: 0.8

5. Convolutional Neural Network

O retea neuronala convolutionala reprezinta un tip de retea neuronala care contine cel putin un strat convolutional pe langa straturile clasice complete. Un strat convolutional bidimensional este alcatuit dintr-un numar de filtre ce reprezinta matrice bidimensionale, prin intermediul carora sunt generate *feature map-uri*. Filtrele sunt apoi aplicate unor tensori de intrare, calculandu-se produsul scalar dintre filtru si regiunea pe care acesta se aplica. Principalele carateristici ale unui filtru sunt: dimensiunea (*kernel size*), distanta unei deplasari pe tensor

(*stride*), functia de activare (*activation function*) si tipul de bordare al tensorului (*padding*). Un alt concept specific acestui tip de retea este stratul de (*pooling*) ce are ca rol reducerea dimensiunilor datelor. Doua astfel de straturi des intalnite sunt (*Max Pooling*) si (*Average Pooling*). De asemenea, o arhitectura clasica contine la final straturi dense, clasice.

Configuratia solutiei finale:

Preprocesare:

- Normalizarea seturilor de date prin impartirea valorilor pixelilor imaginilor la 255.
- Aplicarea codificarii de tip (*one-hot encoding*) etichetelor imaginilor.

Arhitectura modelului:

- Strat convolutional cu 32 de filtre de dimensiuni 3x3 si *padding = "same"*
- Functia de activare LeakyRelu (cu parametrul alpha avand valoarea default 0.3)
- Strat convolutional cu 64 de filtre de dimensiuni 3x3 si *padding = "same"*
- Strat de *Batch Normalization* cu rolul de a standardiza inputul pentru un strat ascuns la fiecare *batch*. Acest strat se poate aplica inainte sau dupa functia de activare dar a dat rezultate mai bune cand a fost aplicat inaintea acesteia.
- Functia de activare LeakyRelu
- Strat de Max Pooling de dimensiune 2x2 si *padding = "same"*
- Strat convolutional cu 64 de filtre de dimensiuni 4x4 si *padding = "same"*
- Strat de *Batch Normalization*
- Functia de activare LeakyRelu
- Strat convolutional cu 128 de filtre de dimensiuni 4x4 si *padding = "same"*
- Strat de *Batch Normalization*
- Functia de activare LeakyRelu
- Strat de Max Pooling de dimensiune 2x2 si *padding = "same"*
- Strat convolutional cu 128 de filtre de dimensiuni 3x3 si *padding = "same"*
- Strat de *Batch Normalization*
- Functia de activare LeakyRelu
- Vectorizarea rezultatului prin functia *Flatten*
- Strat dens de dimensiune 128
- Functia de activare LeakyRelu
- Strat final de 9 neuroni, cu functia de activare *Softmax*

Modelul a fost antrenat pe 30 de epoci si un *batch size = 512* (numarul de exemple de antrenare folosite intr-o singura iteratie). Functia de pierdere folosita este *Categorical Cross-Entropy*

iar algoritmul de optimizare folosit este *Adam - Adaptive Moment Estimation*, o alternativa a algoritmului *Stochastic Gradient Descent*.

Matricea de confuzie:

```
[[468  15   6   5  26   1   8  11  30]
 [  5 498   5   2   3   2   1   5   6]
 [  7  12 455  10  18  18   2   9   2]
 [  7   4   5 498  19  13   3  13  16]
 [ 14   8  13  11 472  15   0  11  10]
 [  2   2  10  11   9 511   4  10   2]
 [ 15   4   3   3   5   6 514   6  24]
 [  8  17  16  13   5  12   4 438   7]
 [  6   3   0   4   2   3   7   1 551]]
```

Configuratia prezentata a obtinut un scor final pe setul de test de **0.89786**. De asemenea, configuratiile testate care au obtinut scoruri mai mici sunt incluse in fisierele predate.