Documentație proiect

1. Implementare

Mașinile cu vectori suport (SVM) reprezintă un model de învățare supervizată (supervised learning) folosit pentru probleme de clasificare și regresie. Pentru a clasifica datele aparținând mai multor clase, există două tipuri de abordări: one vs. all unde sunt antrenați atâți clasificatori câte clase sunt, one vs. one unde sunt antrenați $\frac{nrClase*(nrClase-1)}{2}$ clasificatori. Există 2 tipuri de algoritmi SVM pentru clasificare:

- "hard margin" în care clasificatorul nu este dispus să clasifice greșit;
- "soft margin" prin care i se permite modelului să clasifice greșit câteva date. Acest tip de algoritm este folosit în rezolvarea problemei propuse.

În rezolvarea problemei de clasificare propuse am folosit acest model furnizat de biblioteca ScikitLearn din Python (**from sklearn import svm**). Implementarea din această bibliotecă utilizează abordarea one vs. one.

Alte module / librării importate:

- **numpy**
- **matplotlib.image**, folosit pentru citirea imaginilor
- **from sklearn import preprocessing**, folosit pentru normalizarea datelor
- **from sklearn import metrics**, folosit pentru generarea matricei de confuzie

a) Citirea imaginilor

Cu ajutorul **matplotlib.image** am citit imaginile de antrenare, validare și test cu funcția **imread** și le-am memorat într-o listă pe care am convertit-o într-un **np.array** pe care l-am redimensionat cu funcția **reshape**, transformându-l din 3D în 2D.

b) Normalizarea datelor

Am folosit tot din biblioteca ScikitLearn, **preprocessing** și **StandardScaler** pentru a standardiza datele, utilizând valorile implicite ale parametrilor, anume **copy=True**, care, dacă ar fi fost False, ar fi încercat evitarea unei copii, **with_mean=True** (centrează datele înainte de scalare) și **with_std=True** (scalare la deviația standard).

c) Definirea clasificatorului

Definiția completă:
sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape ='ovr', break_ties=False, random_state=None), unde:

- **C** : un parametru de penalitate pentru eroare care va sugera cât de mult este dispus modelul să evite clasificarea greșită a datelor de antrenare. Dacă C este prea mare, se poate ajunge la overfitting, iar dacă este prea mic, modelul nu va fi capabil să învețe și se poate ajunge la underfitting. Cu alte cuvinte, C se comportă ca un parametru de regularizare;
- **kernel** : o funcție matematică folosită de algoritmii SVM. Poate fi "**rbf**", "**linear**", "**poly**", "**sigmoid**", "**precomputed**" etc. Kernel-ul "**rbf**" este cel mai utilizat și cel mai comun.

Restul parametrilor sunt folosiți cu valorile lor implicite în rezolvarea problemei.

Pentru definirea clasificatorului am folosit : **svm.SVC(C=3, kernel= 'rbf')**. Algoritmul SVM poate dura destul de mult la antrenare: cu cât C este mai mare și funcția kernel mai complexă, cu atât procesul durează mai mult.

d) Antrenarea modelului

Sintaxă: **classifier.fit(train_data,train_labels**), unde **fit** este metoda care va antrena modelul în concordanță cu datele de training primite.

e) Clasificarea datelor de test

Sintaxă: **classifier.predict(test_data)**, unde **predict** este o metoda care va intoarce categoria in care a clasificat fiecare element din test_data.

f) Rezultate:

- Folosing sintaxa **classifier.score(validation_data, validation_labels)** pentru datele de validare am obținut o acuratețe a clasificării de **0.7578**;
- Pentru prima parte unde s-au folosit 25% dintre datele de test, am obținut o acuratețe de **0.76160**;
- La finalul competiției, rezultatul final este de **0.76586**.

2. Matricea de confuzie pentru datele de validare

Fiecare element C_{ij} al matricei de confuzie C reprezintă numărul de elemente din clasa i care au fost clasificate ca fiind din clasa j. Matricea de mai jos este obținută cu ajutorul metodei **metrics.confusion_matrix(validation_labels, validation_predictions)**, unde **validation_labels** reprezintă categoriile corecte ale imaginilor de validare, iar **validation_predictions** reprezintă rezultatul metodei **predict** pentru datele de validare.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
0	361	19	17	15	44	4	30	43	37
1	25	424	10	11	10	5	9	25	8
2	16	24	390	17	32	28	4	18	4
3	28	13	16	423	18	22	16	24	18
4	35	19	23	27	397	11	3	25	14
5	9	6	15	21	13	470	5	20	2
6	25	12	10	11	7	5	474	6	30
7	35	13	18	26	25	14	9	369	11
8	27	4	4	7	3	7	36	8	481

3. Alte încercări de rezolvare a problemei

Clasificatorul Naive Bayse este o tehnică de clasificare bazată pe Teorema lui Bayse din probabilități și este unul dintre cele mai simpli algoritmi de clasificare.

Librării / module importate:

- **numpy**
- **from sklearn.naive_bayes import MultinomialNB**
- **from sklearn import metrics**, folosit pentru generarea matricei de confuzie

a) Detalii de implementare:

- Citirea și memorarea datelor în **np.array**-uri și redimensionarea lor din 3D în 2D, analog implementării metodei SVM;

- Folosirea funcției **linspace** din numpy (**linspace (start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)**) care va împărți intervalul delimitat de **start** și **stop** în **num**-1 intervale egale, adică va returna un array cu **num** numere din interval egal depărtate între ele (diferențele dintre oricare 2 numere consecutive vor fi egale): **bins = np.linspace(0, 255, 9)**;

- Definirea unei functii **values_to_bins(x, bins)** care, pentru fiecare element din array-ul primit ca argument x, întoarce indicele intervalului din bins în care se află cu ajutorul funcției **digitize(x, bins, right=False)** din numpy, unde **right** este un

parametru opțional ce semnifică dacă capătul drept al intervalului este sau nu luat în considerare:
x_train = values_to_bins(train_data_reshaped, bins)
x_validation = values_to_bins(validation_data_reshaped, bins)
x_test = values_to_bins(test_data_reshaped, bins)

- Definirea modelului: **naive_bayes_model = MultinomialNB();**
- Antrenarea modelului cu ajutorul metodei **fit** și a datelor de training: **naive_bayse_model.fit(x_train, training_labels);**
- Clasificarea imaginilor de validare pentru a putea calcula matricea de confuzie: **naive_bayse_model.predict(validation_data).**

b) Rezultat:

- Pentru datele de validare, cu ajutorul metodei **score**, am obținut acuratețea **0.1108**.

c) Matricea de confuzie:

- Matricea de confuzie pentru datele de validare, obținută, de asemenea, cu **metrics.confusion_matrix(validation_labels, validation_predictions)**:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
0	0	0	0	0	0	570	0	0	0
1	0	0	0	0	0	527	0	0	0
2	0	0	0	0	0	533	0	0	0
3	0	0	0	0	0	578	0	0	0
4	0	0	0	0	0	554	0	0	0
5	0	0	0	0	0	561	0	0	0
6	0	0	0	0	0	580	0	0	0
7	0	0	0	0	0	520	0	0	0
8	0	0	0	0	0	577	0	0	0