Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2019-2020

Artificial Intelligence (AI) applications for COVID-19 pandemic

Name: Pacurar Cristian
Group: 30233
Email: pacurarcristian31@gmail.com

Contents

Chapter 1

Abstract

În această criză mondială de sănătate, industria medicală cauta noile tehnologii de monitorizare și control al răspândirii COVID- 19 (Coronavirus). AI este una dintre aceste tehnologii care poate urmări cu ușurință răspândirea acestui virus, identifică pacienții cu risc ridicat, și este utilă în controlul acestei infecții în timp real. Poate deasemenea prezice riscul de mortalitate prin analiza adecvată a datelor anterioare a pacientilor. AI ne poate ajuta să combatem acest virus cu ajutor medical, notificare și sugestii despre controlul infecției.Această tehnologie are potențialul de a îmbunătăți planificarea, tratamentul și rezultatele raportate ale pacientului infectat cu COVID-19.

Diagrama de flux de mai jos informează și compară fluxul de tratament minim non-AI comparativ cu tratamentul pe bază de AI. Diagrama de flux explică implicarea AI în etapele semnificative de tratament cu o precizie ridicată și reduce complexitatea și timpul necesar. Medicul nu se concentrează numai pe tratamentul pacient, dar și controlul bolii cu aplicația AI. Simptomele majore și analiza testelor se fac cu ajutorul AI cu cea mai mare precizie. De asemenea, arată că reduce numărul total de pașii făcuți în întregul proces.

Figure 1.1: Covid-19

Chapter 2

Detalii de implementare

1. Datele pentru acest test au fost luate de pe site-ul www.kaggle.com. Vom folosi ca si date de antrenare 100 de celule din tabelul anexat, iar ca date de test ultimele 21 zile din tabelul de test. In total sunt 121 de celule. Dupa ce vom reusi sa obtinem un sistem care sa fie suficient de precis, vom incerca sa facem predictii pentru viitor, de exemplu sa vedem cum va arata situatia in luna octombrie 2020, cand va reincepe un nou an universitar.
   Setul de date a fost ales prin selectarea cazurilor care apartin strict Romaniei.
   Sursa: `https://www.kaggle.com/ashudata/covid19dataset#COVID_Data.csv`

	No.	Country	Date	Confirmed	Death	newConfirmed	newDeath
0	1	Romania	2019-12-31	0	0	0	0
1	2	Romania	2020-01-01	0	0	0	0
2	3	Romania	2020-01-02	0	0	0	0
3	4	Romania	2020-01-03	0	0	0	0
4	5	Romania	2020-01-04	0	0	0	0

Figure 2.1: Data set

- Coloana No. reprezinta numarul raportului, coloana country reprezinta tara de referinta, Coloana Date reprezinta data la care a fost dat raportul, coloana confirmed reprezinta numarul de cazuri confirmate pana la data respectiva, coloana death reprezinta numarul de persoane decedate pana la data respectiva, iar coloanele newConfirmed si newDeath reprezinta numarul de cazuri noi aparute respectiv numarul de persoane decedate strict in acea zi.

- Acest set de date etichetate a fost ales pentru ca este precis si pentru ca reprezinta o sursa sigura de informatii reale.

2. Explicarea alegerii metodei de supervised machine learning.

Metoda de supervised learning machine se potriveste excelent cu problema propusa. Dimensiunea setului de date este relativ redusa si are date strict numerice sau de tip data calendaristica. Aceasta metoda ne va permite sa invatam sistemul pe un set de antrenare concret oferit de noi, iar dupa aceea, sistemul nu va trebui decat sa prezica corect cateva date de test ce nu au fost date in procesul de invatare.

3. Explicarea alegerii regresiei.

   Prin definitie, modelul probabilistic prin regresie este creat pentru a aproxima mapping functions (f) din variabile de intrare (X) catre un set continuu de variabile de iesite (Y). Practic, vom avea de aproximat evolutia unei functii.

Chapter 3

Algoritmi de machine learning

1. NN - Multi-Layer Perceptron

   (a) Multi-Layer Perceptron (MLP) este o clasa de retea neuronala artificiala avansata (ANN). Un MLP este format din cel putin trei straturi de noduri: un strat de intrare, un strat ascuns si un strat de iesire. Cu exceptia nodurilor de intrare, fiecare nod este un neuron care utilizeaza o functie de activare neliniara. MLP utilizeaza o tehnica de invatare supravegheata numita backpropagation pentru instruire. Straturile sale multiple si activarea neliniara disting MLP de un perceptron liniar. Poate distinge date care nu sunt liniar separabile.

   (b) Clasa MLPClassifier implementeaza un algoritm MLP care se antreneaza utilizand Backpropagation. MLP se antreneaza pe doua tablouri: matricea X de dimensiune (n_samples, n_features), care are date de training reprezentate ca vectori flotanti si matricea y de dimensiune (n_samples), care are ca valori etichete de clasa pentru training.
   Clasa MLPRegressor pune in aplicare un MLP care se antreneaza folosind backpropagation fara nicio functie de activare in stratul de iesire, folosind functia de identitate ca functie de activare. Prin urmare, utilizeaza square error ca functie de pierdere, iar iesirea este un set de valori continue.

   (c) Datele de test reprezinta 17 la suta din totalul de date, acest lucru facandu-se direct din cod prin argumentul test_size = 0.17 al functiei train_test_split, astfel restul datelor ramanand pentru antrenarea algoritmului.

	Confirmed	Death	newConfirmed	newDeath
24	0	0	0	0
103	6300	316	310	25
93	2738	115	278	23
67	9	0	0	0
92	2460	92	215	10
...	...	...	...	...
91	2245	82	136	17
109	8418	421	351	10
29	0	0	0	0
13	0	0	0	0
23	0	0	0	0

Figure 3.1: Date de training

```
y_train, y_test, X_train, X_test = train_test_split(X_final, y, test_size=0.17)
```

Figure 3.2: Splitting the data

(d) Am folosit MLPRegressor cu diferiti parametrii (ex: hidden_layer_sizes, solver etc.), reusind astfel sa imbunatatim performanta algoritmului si sa observam diferentele rezultate.

```
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

#folosim MLP regressor pentru predictii
#MLP Regressor pentru NN Multy Layer
regr = MLPRegressor(solver='adam', hidden_layer_sizes=(200,100), max_iter=10000, verbose = 'true',activation='relu'

#Antrenam modelul folosind setul de train
regr.fit(X_train, y_train)

#Facem predictii folosind setul de test
y_pred = regr.predict(X_test)
```

```
Iteration 35, loss = 1549840.46627854
Iteration 36, loss = 1546238.71573484
Iteration 37, loss = 1542551.17086318
Iteration 38, loss = 1538754.39791020
Iteration 39, loss = 1534850.95186319
Iteration 40, loss = 1530807.02595050
Iteration 41, loss = 1526648.23437153
Iteration 42, loss = 1522368.28930020
Iteration 43, loss = 1517962.80080491
Iteration 44, loss = 1513430.10616110
Iteration 45, loss = 1508774.46759529
Iteration 46, loss = 1503977.43659328
Iteration 47, loss = 1499053.71830002
Iteration 48, loss = 1493997.56759435
Iteration 49, loss = 1488807.50785482
Iteration 50, loss = 1483482.72343469
Iteration 51, loss = 1478022.56170012
Iteration 52, loss = 1472426.53427457
Iteration 53, loss = 1466694.29461367
Iteration 54, loss = 1460825.60486709
```

Figure 3.3: Results

(e) Am normalizat, standardizat si scalat setul de date cu ajutorul StandardScaler si MinMaxScaler. Am aplicat aceste operatii mai intai pe setul de training, iar mai apoi pe setul de testing. Dupa testare am observat ca mean squared error este mai mare.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

#Folosim diferite scalere pentru a puteam face operatii de normalizare, standardizare si scalare
std_scaler = StandardScaler()
std_scaler2 = StandardScaler()
minMaxScaler = MinMaxScaler()
```

```
minMaxScaler.fit(X_train)
X_train = minMaxScaler.transform(X_train)

#Normalizam si scalam setul de train
std_scaler.fit(X_train)
X_train = std_scaler.transform(X_train)
std_scaler2.fit(X_train)
X_train = std_scaler2.transform(X_train)
```

```
#Normalizam si scalam setul de test
minMaxScaler.fit(X_test)
X_test = minMaxScaler.transform(X_test)
std_scaler.fit(X_test)
X_test = std_scaler.transform(X_test)
```

```
#Scalam si standardizam datele de test
std_scaler2.fit(X_test)
X_test = std_scaler2.transform(X_test)
```

```
#Antrenam modeul folosind setul de train
regr.fit(X_train, y_train)

#Facem predictii
y_pred = regr.predict(X_test)
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
```

Figure 3.4: Results

(f) Cum nu avem date continue in setul de date am fost nevoit sa folosesc LabelEncoder pentru coloana Country si coloana Date. Astfel datele din string s-au transformat in int, fiecare valoare din tabelul initial fiind reprezentata de un numar sau eticheta unica.

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#folosim un label encoder pentru coloanele country si date pentru a le putea transforma in valori numerice
le = preprocessing.LabelEncoder()

le.fit(data['Date'])
data.loc[:, 'Date'] = le.transform(data['Date'])

le.fit(data['Country'])
data.loc[:, 'Country'] = le.transform(data['Country'])
```

Figure 3.5: LabelEncoder use

(g) Pentru acuratete am folosit cross_validation_score cu argumentul cv = 4.

De asemenea, am folosit explained_variance_score, max_error, mean_absolute_error, mean_squared_error, median_absolute_error si r2_score.

```
import warnings
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor

warnings.filterwarnings('ignore')

#MLP Regressor pentru NN Multy Layer
regr = MLPRegressor(solver='adam', hidden_layer_sizes=(200,100), max_iter=10000,activation='relu')

#Cross validation score
cross_val_score(regr, y_test, X_test, cv=4)
```

Figure 3.6: Cross validation

```
#Am folosit diferite metrici pentru a putea estima acuritatea rezultatelor obtinute
from sklearn.metrics import explained_variance_score
explained_variance_score(y_test, y_pred)
```

```
0.9989264372755813
```

```
from sklearn.metrics import max_error
max_error(y_test, y_pred)
```

```
298.09753935656954
```

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

```
60.95082109273548
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

```
11038.275925048712
```

```
from sklearn.metrics import mean_squared_log_error
mean_squared_log_error(y_test, y_pred)
```

```
2.7232104926734797
```

```
from sklearn.metrics import median_absolute_error
median_absolute_error(y_test, y_pred)
```

```
15.534019723698975
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.9988518416658613
```

Figure 3.7: Metrici de evaluare

Explained Variance Score: functie explicativa de scadere a variatiei regresiei. Cel mai bun scor posibil este 1. Max Error: calculeaza rezidul maxim al erorii. Cea mai buna valoare posibila este 0. Mean Absolute Error: eroarea absoluta a pierderii regresiei. Cea mai buna valoarea e 0. Mean Squared Error: eroarea la patrat a pierderii regresiei. Cea mai buna valoare e 0. Mean Squared Log Error: eroare logartimica la patrat a pierderii regresiei. Cea mai buna valoare e 0. Median Absolute Error: pierdere de regresie absoluta medie. Cea mai buna valoare e 0. R2 Score: functia scorului de regresie. Cea mai buna valoare este 1.

	Confirmed	Death	newConfirmed	newDeath
42	0	0	0	0
41	0	0	0	0
120	11978	693	362	30
104	6633	331	333	15
44	0	0	0	0
65	6	0	2	0
100	5202	248	441	28
80	308	0	31	0
61	3	0	0	0
57	1	0	1	0
114	10096	545	386	21
111	8936	478	190	27
96	3864	151	251	5
59	3	0	2	0
18	0	0	0	0
69	15	0	0	0
101	5467	270	265	22
50	0	0	0	0
60	3	0	0	0
106	7216	372	337	21
56	0	0	0	0

Figure 3.8: Expected

(h) Imbunatatirea rezultatelor folosind diferiti parametri

NN parametrii schimbati:
NN: MLPRegressor(solver='adam', hidden_layer_sizes=(200,100), max_iter=10000, activation='relu')
NN_2: MLPRegressor(solver='lbfgs', hidden_layer_sizes=(200,150), max_iter=20000, activation='relu', alpha = 0.0003)
NN_3: MLPRegressor(solver='adam', hidden_layer_sizes=(200,100), max_iter=10000, activation='relu')

In urma schimbarii parametrilor, rezultatele au fost aproximativ la fel de bune sau proaste. In unele cazuri la diferite metrici au fost cu putin mai bune insa cu o diferenta neglijabila. Deoarece datele de test si de training se iau aleator din setul de date, rezultatele metricilor vor diferii la fiecare rulare. Am realizat ca setul de date nu este suficient de mare pentru a face o diferenta semnificativa in urma alternarii parametrilor metodelor de regresie. Insa totusi diferiti parametrii pot optimiza anumite aspecte ale algoritmului, insa se pot testa doar pe un numar finit de combinatii ale acestora.

(i) Reprezentari grafice

```
import matplotlib.pyplot as plt
data.hist()
plt.show()
```

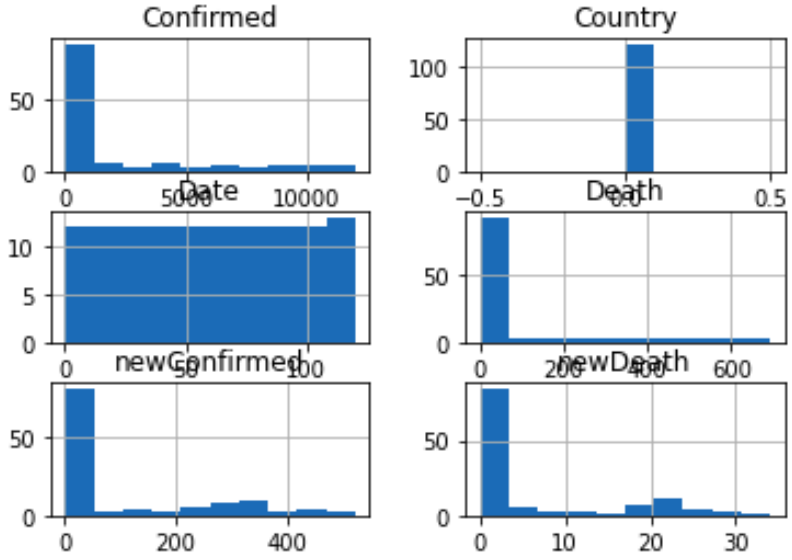

Figure 3.9: Histograma

```
plt.scatter(data['Date'], data['Death'])
plt.show()
```

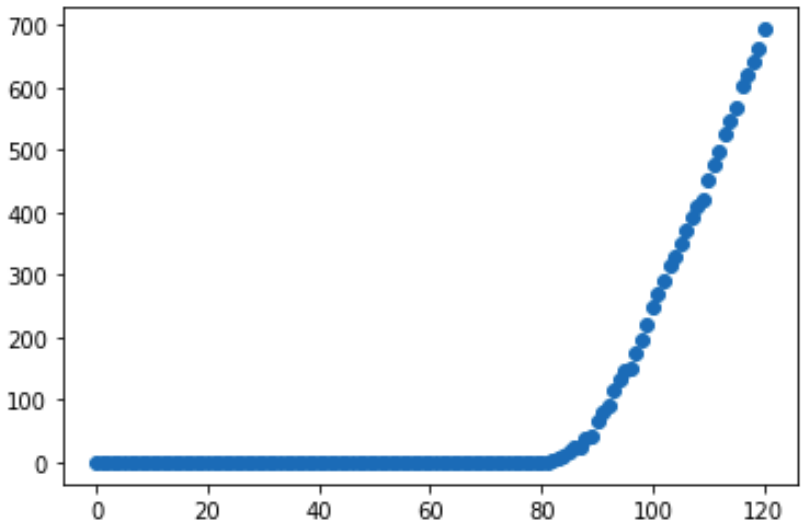

Figure 3.10: Evolutie Death/Date

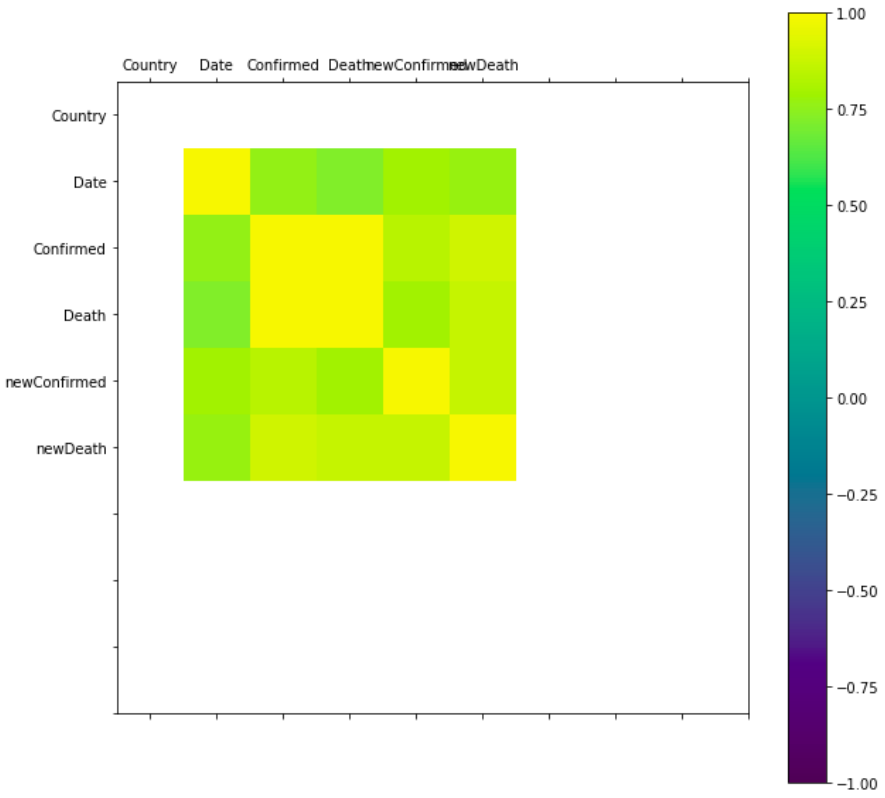

Figure 3.11: Correlation matrix

O matrice de corelație este un tabel care prezintă coeficienții de corelație între variabile. Fiecare celulă din tabel arată corelația dintre două variabile. O matrice de corelație este utilizată pentru a rezuma datele, ca o intrare într-o analiză mai avansată și ca un diagnostic pentru analize avansate.

Chapter 4

Concluzie

Inteligența artificială este un instrument de viitor și utila la identificarea infecților precoce cauzate de coronavirusului și, de asemenea, ajută la monitorizarea starii pacienților infectați. Se poate îmbunătăți semnificativ coerența tratamentului și luarea deciziilor prin dezvoltarea utilă de algoritmi. AI nu este utila doar în tratamentul pacienților infectați de COVID-19, ci și pentru monitorizarea corectă a sănătății lor.

Aceasta poate urmări criza COVID-19 la diferite scale, cum ar fi cele medicale, aplicații moleculare și epidemiologice. De asemenea, este utila in facilitarea cercetarii acestui virus, utilizând analiza datelor disponibile. AI poate ajuta la dezvoltarea regimurilor de tratament adecvate, strategii de viitor, dezvoltarea de medicamente și vaccinuri.

Bibliography

[1] Multilayer perceptron,
https://en.wikipedia.org/wiki/Multilayer_perceptron

[2] SkLearn bib,
https://scikit-learn.org/stable/

[3] Advantages and disadvantages for algorithms,
https://medium.com