

# **TEMA 1**

## **REȚELE NEURONALE ARTIFICIALE**

**STUDENT: Vasile Andrei – Daniel**

**SPECIALIZAREA: Informatică**

**ANUL DE STUDII: III**

**GRUPA: 40322**

**DISCIPLINA: Inteligență Artificială**

**TITULAR CURS: Conf. dr. Simona Nicoară**

**TITULAR LABORATOR: Lector dr. Elia Dragomir**

## Cuprins

1. Descrierea problemei și a bazei de date.....	2
2. Variabila dependentă și variabilele independente .....	4
3. Prelucrarea setului de date – dificultăți și provocări .....	4
4. Justificarea tipului modelului .....	5
5. Parametrii aleși pentru evaluarea modelului.....	5
6. Tabel centralizator al rezultatelor experimentale .....	5
7. Cod sursă.....	6
8. Concluzii finale .....	9

## 1. Descrierea problemei și a bazei de date

Pentru realizarea acestui proiect s-a folosit o bază de date de tipul „Housing Price Dataset”, descărcată de la adresa <https://www.kaggle.com/datasets/yasserh/housing-prices-dataset/data>.

Setul de date are 546 de înregistrări, fiecare înregistrare având 11 parametrii (caracteristici), după cum urmează:

Price	
Descriere	Reprezintă prețului casei
Unitate de măsură	Nu este specificată
Valoare minimă	1750000
Valoare maximă	13300000

Area	
Descriere	Reprezintă aria casei
Unitate de măsură	Nu este specificată
Valoare minimă	1650
Valoare maximă	16200

Bedrooms	
Descriere	Reprezintă numărul de dormitoare ale casei
Unitate de măsură	Număr de elemente
Valoare minimă	1
Valoare maximă	6

Bathrooms	
Descriere	Reprezintă numărul de băi ale casei
Unitate de măsură	Număr de elemente
Valoare minimă	1
Valoare maximă	4

Stories	
Descriere	Reprezintă numărul de etaje ale casei
Unitate de măsură	Număr de elemente
Valoare minimă	1
Valoare maximă	4

Mainroad	
Descriere	Specifică dacă locuința are acces direct la drum principal
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Guestroom	
Descriere	Specifică dacă există o cameră pentru oaspeți
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Basement	
Descriere	Specifică dacă locuința are subsol/beci
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Hotwaterheating	
Descriere	Specifică dacă locuința are sistem de încălzire cu apă caldă
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Airconditioning	
Descriere	Specifică dacă locuința are aer condiționat
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Parking	
Descriere	Specifică numărul de locuri de parcare disponibile pe proprietate
Unitate de măsură	Număr de elemente
Valoare minimă	0
Valoare maximă	3

Prefarea	
Descriere	Specifică dacă locuința este într-o zonă preferențială
Unitate de măsură	Nu este cazul
Mulțimea de valori	Yes, No

Furnishingstatus	
Descriere	Reprezintă starea de mobilare a locuinței
Unitate de măsură	Nu este cazul
Mulțimea de valori	Unfurnished, semi-furnished, furnished

## 2. Variabila dependentă și variabilele independente

S-a încercat construirea unui model care să prezică prețul locuințelor pe baza celorlalte caracteristici. Prin urmare, variabila pe care vrem să o prezicem (variabila dependentă sau rezultatul) este reprezentată de prețul locuințelor (numită „price” în dataset). Factorii care influențează rezultatul (sau variabilele independente) sunt reprezentați de toate celelalte atribute, și anume: area, bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, prefarea și furnishingstatus.

## 3. Prelucrarea setului de date – dificultăți și provocări

Menționez că nu am eliminat niciun atribut din setul de date deoarece am considerat că toate caracteristicile pot influența prețul unei case. Dataset-ul nu a avut valori lipsă, însă așa cum se poate observa și din descrierea anterioară a atributelor, multe dintre acestea nu aveau valori numerice. Pentru a rezolva această problemă am procedat în felul următor. Valorile „yes” din coloanele {"mainroad", "guestroom", "basement", "hotwaterheating", "airconditioning", "prefarea"} au fost înlocuite cu cifra 1 și valorile de „no” au fost înlocuite cu cifra 0.

Coloana furnishingstatus avea 3 valori: unfurnished, semi-furnished și furnished. Nu știu sigur dacă este adevărat dar am înțeles că dacă le-aș fi codificat cu 3 valori numerice diferite (cum ar fi 0, 1 și 2), rețeaua neuronală artificială putea considera că există o ordine numerică de tipul „mic-mediu-mare”, lucru care o influența negativ. Prin urmare s-a folosit metoda One Hot Encoding (OHE), care transformă fiecare categorie separată într-o coloană cu 0 sau 1. În acest fel, toate categoriile sunt tratate în mod egal. În cazul de față prima coloană a fost eliminată pentru a evita redundanța (dacă primele două coloane create – fie ele „semi-furnished” și „unfurnished” au valoarea 0, înseamnă că atributul inițial are cea de-a treia valoare și anume furnished). Așadar, se poate observa că pentru a face diferența între trei categorii, sunt necesare doar două valori.

O altă problemă majoră a fost că valorile atributelor nu erau într-un interval numeric apropiat (de exemplu price-9240000, area-3500, bedrooms-4). Datorită acestor „distanțe mari” dintre valori, este mult mai dificil de antrenat o rețea neuronală artificială astfel încât să fie obținute niște predicții decente. De exemplu, modelul poate ajunge să învețe preponderent pe baza coloanelor cu valori mari și să le ignore pe cele cu valori mici. Deși s-a încercat rezolvarea acestei situații, nu s-a reușit în totalitate. În acest sens s-a recurs la scalarea datelor, adică transformarea valorilor astfel încât să fie pe același interval numeric. Pentru implementarea scalării s-a utilizat metoda **StandardScaler** din Python, metodă ce transformă fiecare coloană astfel încât media valorilor sale să devină 0 și deviația sau abaterea standard să devină 1. Formula matematică pentru fiecare valoare  $x_i$  dintr-o coloană este:

$xi(scaled) = (xi - \mu) / \sigma$  , unde:

- $\mu$  reprezintă media valorilor din coloană
- $\sigma$  reprezintă deviația standard a coloanei

Așa cum s-a menționat și mai sus, după scalare coloanele vor avea media apropiată de valoarea 0 și abaterea standard 1. Deci datele au fost scalate și apoi a fost antrenată rețeaua pe acestea. La sfârșit s-a făcut „unscale” (operația inversă scalării) pe valorile prezise de model pentru a putea fi comparate cu cele reale.

#### 4. Justificarea tipului modelului

Modelul rețelei neuronale este de regresie deoarece are variabila țintă (price) este numerică. Modelul încearcă să prezică un număr real, nu o anumită clasă dintre mai multe posibile. Structura rețelei reflectă și ea acest lucru, ultimul strat având doar un singur neuron fără funcție de activare. Dacă s-ar fi dorit prezicerea unei variabile categoricale, atunci modelul ar fi avut pe ultimul strat un număr de neuroni egal cu mulțimea de valori ale categoriilor respective și ar fi fost de clasificare.

#### 5. Parametrii aleși pentru evaluarea modelului

Cei doi parametri statistici folosiți pentru a evalua rețeaua neuronală artificială sunt Eroarea Medie Pătratică – utilizată ca și Loss - (Mean Squared Error - MSE) și Rădăcina Erorii Pătratice medii (Root Mean Squared Error - RMSE). MSE măsoară media pătratelor erorilor, mai exact diferențele dintre valorile prezise și cele reale. Bineînțeles, cu cât aceasta este mai mică cu atât modelul este mai bun. RMSE este calculată de două ori în interiorul buclei K-Fold, odată pe datele scalate și apoi pe datele originale, cea din urmă valoare reprezentând performanța reală a modelului. Aceasta (RMSE) mai este calculată odată la sfârșitul antrenării, pentru a determina scorul real al întregului proces de validare încrucișată.

#### 6. Tabel centralizator al rezultatelor experimentale

Rezultate experimentale pentru RNA de regresie						
Nume Rețea	Stratul ascuns H1	Stratul ascuns H2	Stratul Ascuns H3	Epoci	RMSE	Eroare / loss
RNA 1	25	-	-	100	1133792.8735680967	0.2333
RNA 2	18	-	-	300	1257256.778876266	0.2516

RNA 3	8	20	-	100	1126204.5974058814	0.2425
RNA 4	28	20	-	100	1124555.715458942	0.2189
RNA 5	10	6	-	300	1134373.155645393	0.2244
RNA 6	50	25	-	200	1117243.6547391368	0.2283
RNA 7	18	22	-	300	1114741.552595581	0.1836
RNA 8	10	6	8	500	1117502.8430968705	0.2835
RNA 9	4	28	30	200	1148066.9192547589	0.2222
RNA 10	8	10	4	100	1224751.9642651565	0.2846
RNA 11	16	16	16	400	1149132.2773859878	0.2250
RNA 12	12	24	48	200	1168550.5708894476	0.2394

Rezultate experimentale pentru RNA de regresie							
Nume Rețea	Stratul ascuns H1	Stratul ascuns H2	Stratul Ascuns H3	Stratul Ascuns H4	Epoci	RMSE	Eroare / loss
RNA 13	5	10	10	5	200	1119788.921495609	0.2843
RNA 14	8	12	24	50	100	1147639.4942252452	0.2132
RNA 15	30	17	10	2	300	1240982.395846265	0.3435
RNA 16	15	30	12	40	500	1104816.3069104075	0.1724

## 7. Cod sursă

```

1 import pandas as pd
2 import os
3 import numpy as np
4 from sklearn import metrics
5 from keras import layers
6 from scipy.stats import zscore
7 from sklearn.model_selection import KFold
8 from keras.models import Sequential
9 from keras.layers import Dense, Activation
10 from keras.callbacks import EarlyStopping
11 from sklearn.preprocessing import StandardScaler
12 from sklearn.model_selection import train_test_split
13 import matplotlib.pyplot as plt

```

```

1 path = "./data/"
2 filename_read = os.path.join(path, "housing.csv")
3 filename_write = os.path.join(path, "housing-out.csv")
4 df = pd.read_csv(filename_read, na_values=['NA', '?'])
5

```

```

1 # replace the values yes or no with 0 and 1
2 cols_yes_no = ["mainroad", "guestroom", "basement", "hotwaterheating", "airconditioning", "prefarea"]
3 for col in cols_yes_no:
4     df[col] = df[col].map({"yes":1, "no":0})
5

```

```

1 if 'furnishingstatus' in df.columns:
2     df = pd.get_dummies(df, columns=['furnishingstatus'], drop_first=True)
3
4 # because the resulting columns are of type bool, we convert them to int
5 bool_cols = ['furnishingstatus_semi-furnished', 'furnishingstatus_unfurnished']
6 df[bool_cols] = df[bool_cols].astype(int)

```

```

1 # Schuffle the dataset
2 np.random.seed(42)
3 df = df.reindex(np.random.permutation(df.index))
4 df.reset_index(inplace=True, drop=True)
5
6 dataset = df.values

```

```

1 x = dataset[:,1:14]
2 y = dataset[:,0]
3
4 # Cross-Validate
5 kf = KFold(5)
6
7 oos_y = []
8 oos_pred = []
9 fold = 0
10
11 # KFold.split(x) returns indices, that is the rows in the dataset that are part of the train and test for the current fold
12 for train_idx, test_idx in kf.split(x): #train_idx and test_idx are lists of indices (for clarity I put idx at the end)
13     fold+=1
14     print("Fold #{}".format(fold))
15
16     # Split in train/test
17     x_train, x_test = x[train_idx], x[test_idx]
18     y_train, y_test = y[train_idx], y[test_idx]

```

```

20 # ==SCALE DATA==
21 scaler_x = StandardScaler()
22 x_train_scaled = scaler_x.fit_transform(x_train)
23 x_test_scaled = scaler_x.transform(x_test)
24
25 scaler_y = StandardScaler()
26 y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
27 y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))
28

```

```

29 # CREATE MODEL
30 model = Sequential()
31 model.add(layers.Input(shape=(x_train.shape[1],)))
32 model.add(layers.Dense(15, activation='relu'))
33 model.add(layers.Dense(30, activation='relu'))
34 model.add(layers.Dense(12, activation='relu'))
35 model.add(layers.Dense(40, activation='relu'))
36 model.add(layers.Dense(1))
37 model.compile(loss='mean_squared_error', optimizer='adam')
38
39 monitor = EarlyStopping(monitor='val_loss', min_delta=1e-5, patience=8, verbose=1, mode='auto')

```

```

44 # PREDICT
45 pred_scaled = model.predict(x_test_scaled)
46
47 # Unscale and save results
48 pred_real = scaler_y.inverse_transform(pred_scaled)
49 y_real = scaler_y.inverse_transform(y_test_scaled)
50 oos_y.append(y_real)
51 oos_pred.append(pred_real)
52
53
54 # === Measure this fold's RMSE ===
55 # Scikit-learn expects both arrays to have the same shape, and pred_real is a 2D array of the shape (num_samples, 1).
56 # So we make y_test a 2D array as well -> y_test.reshape(-1, 1)
57 rmse_scaled = np.sqrt(metrics.mean_squared_error(pred_scaled, y_test_scaled))
58 print("Fold score (RMSE, scaled): {}".format(rmse_scaled))
59
60 score = np.sqrt(metrics.mean_squared_error(pred_real, y_real))
61 print("Fold score (RMSE): {}".format(score))

```

```

65 # Build the oos prediction list and calculate the error.
66 oos_y = np.concatenate(oos_y)
67 oos_pred = np.concatenate(oos_pred)
68 score = np.sqrt(metrics.mean_squared_error(oos_pred, oos_y))
69 print("Final, out of sample score (RMSE): {}".format(score))
70
71
72
73 # Write the cross-validated prediction in csv
74 # Flatten 2D arrays into 1D arrays
75 oos_y_flat = np.concatenate(oos_y).flatten()
76 oos_pred_flat = np.concatenate(oos_pred).flatten()
77 # Convert to DataFrames
78 oos_y_df = pd.DataFrame(oos_y_flat, columns=['RealPrice'])
79 oos_pred_df = pd.DataFrame(oos_pred_flat, columns=['PredictedPrice'])
80 # Concatenate with original df
81 oosDF = pd.concat([df.reset_index(drop=True), oos_y_df, oos_pred_df], axis=1)
82 # Save to CSV
83 oosDF.to_csv(filename_write, index=False)

```

```

1
2 def chart_regression(pred, y, sort=True):
3     t = pd.DataFrame({'pred': pred, 'y': y.flatten()})
4     if sort:
5         t.sort_values(by=['y'], inplace=True)
6     plt.plot(t['y'].tolist(), label='expected')
7     plt.plot(t['pred'].tolist(), label='prediction')
8     plt.ylabel('output')
9     plt.legend()
10    plt.show()
11
12
13 # Plot the chart
14 chart_regression(oos_pred.flatten(), oos_y.flatten())

```

## 8. Concluzii finale

În urma efectuării experimentelor, putem spune că dacă urmărim obținerea valori pentru loss (MSE) și RMSE cât mai mici, pentru acest set de date, în condițiile de experimentare date, RNA 16 este cea mai potrivită arhitectură pentru regresie, având următoarea structură:

- Hidden Layer 1: 15 neuroni, activare ReLU
- Hidden Layer 2: 30 neuroni, activare ReLU
- Hidden Layer 3: 12 neuroni, activare ReLU
- Hidden Layer 4: 40 neuroni, activare ReLU

Alți parametri care au fost utilizați:

- Funcția de cost: Mean Squared Error (MSE)
- Optimizator: Adam
- Numărul de epoci: maxim 500, dar în practica între 13 și 50, în funcție de early stopping

Trebuie menționat că numărul în care am împărțit dataset-ul (KFold) a avut mereu valoarea 5. Ținând cont că dataset-ul are 546 de înregistrări, de fiecare dată au fost folosite aproximativ 436 pentru partea de antrenare și 109 pentru testare.

Valoarea RMSE pentru toate valorile din partea de antrenare a fost 1104816.3069104075, iar ultimul loss-ul pentru datele de antrenare a fost 0.1724.