

Documentație FeedbackHHC

Link repository:

[CurcudelTeodor/FeedbackHHC \(github.com\)](https://github.com/CurcudelTeodor/FeedbackHHC)

Introducere

Scopul proiectului este sa prezica feedback-ul dat de pacienți unor agenții de sanatate folosind algoritmi de Machine Learning (diferiți clasificatori, rețele neuronale). Acești algoritmi vor “descoperi” cele mai relevante attribute (caracteristici) ale unui pacient și le vor folosi pentru a face predicția.

Preprocesarea datelor

Setul de date inițial are multe valori lipsă și coloane irelevante. Pașii pe care i-am făcut pentru a avea un set de date consistent au fost următorii: ștergerea instanțelor care au null-uri pe o coloana, ștergerea coloanelor care încep cu Footnote și coloanei Address (fiind irelevante pentru scopul proiectului).

```
columns_to_drop = [col for col in data.columns if col.startswith('Footnote')]
data = data.drop(columns=columns_to_drop)
data = data.drop(columns=['Address'])
```

Apoi am codificat coloanele cu valori întregi și am transformat Certification Date într-un timestamp pentru a lucra mai ușor cu ele.

```
data['State'] = data['State'].map(item_to_label('1'))
data['Provider Name'] = data['Provider Name'].map(item_to_label('2'))
data['City/Town'] = data['City/Town'].map(item_to_label('4'))
data['Type of Ownership'] = data['Type of Ownership'].map(item_to_label('5'))
data['Certification Date'] = data['Certification Date'].map(date_to_timestamp)
```

După acești pași, toate valorile erau de tip Object deci trebuia sa le transformăm în tipul lor corect.

```
# 162 -> int
# 1,241 -> int
# 1.643 -> float
# Same As National string -> int8 (we convert it into number 1)
```

Apoi am codificat coloanele categorice pentru a lucra mai ușor cu ele.

```
categorization_mapping = {
    'Worse Than National Rate': 0,
    'Same As National Rate': 1,
    'Better Than National Rate': 2
}

categorization_mapping_yes_no = {
    'No': 0,
    'Yes': 1,
}
```

Apoi am înlocuit toate celulele NA cu valoarea mediană dacă coloana e cuantificabilă sau cu cea mai frecventă valoare în caz contrar.

```
for col in data.columns:
    data[col] = fill_column(data, col)

def fill_column(data: pd.DataFrame, column_name: str):
    imposter = np.nan
    column = data[column_name]
    filtered_column = column[column != imposter]

    if column_name in QUANTIFIABLE_COLUMNS:
        return column.replace(imposter, filtered_column.median())
    else:
        # get the most frequent value
        return column.replace(imposter, filtered_column.mode().iloc[0])
```

Am făcut și o Analiză pe Componentele Principale (PCA) pentru a reduce numărul de feature-uri pentru o instanță deoarece era dificil de lucrat cu 50 de coloane (feature-uri).

```
def pca_transform(data: pd.DataFrame, target_variance=0.90):
    print(f'Initial data shape: {format(data.shape)}')

    # normalize columns
    for col in data.columns:
        data[col] = (data[col] - data[col].min()) / (data[col].max() -
data[col].min())

    pca = PCA()
    pca.fit(data)

    variance = np.cumsum(pca.explained_variance_ratio_)
    principal_components_count = np.where(variance >= target_variance)[0][0] + 1

    # apply pca to the data
    pca = PCA(n_components=principal_components_count)
    data = pd.DataFrame(pca.fit_transform(data))

    print(f'PCA data shape: {format(data.shape)}')

    return data
```

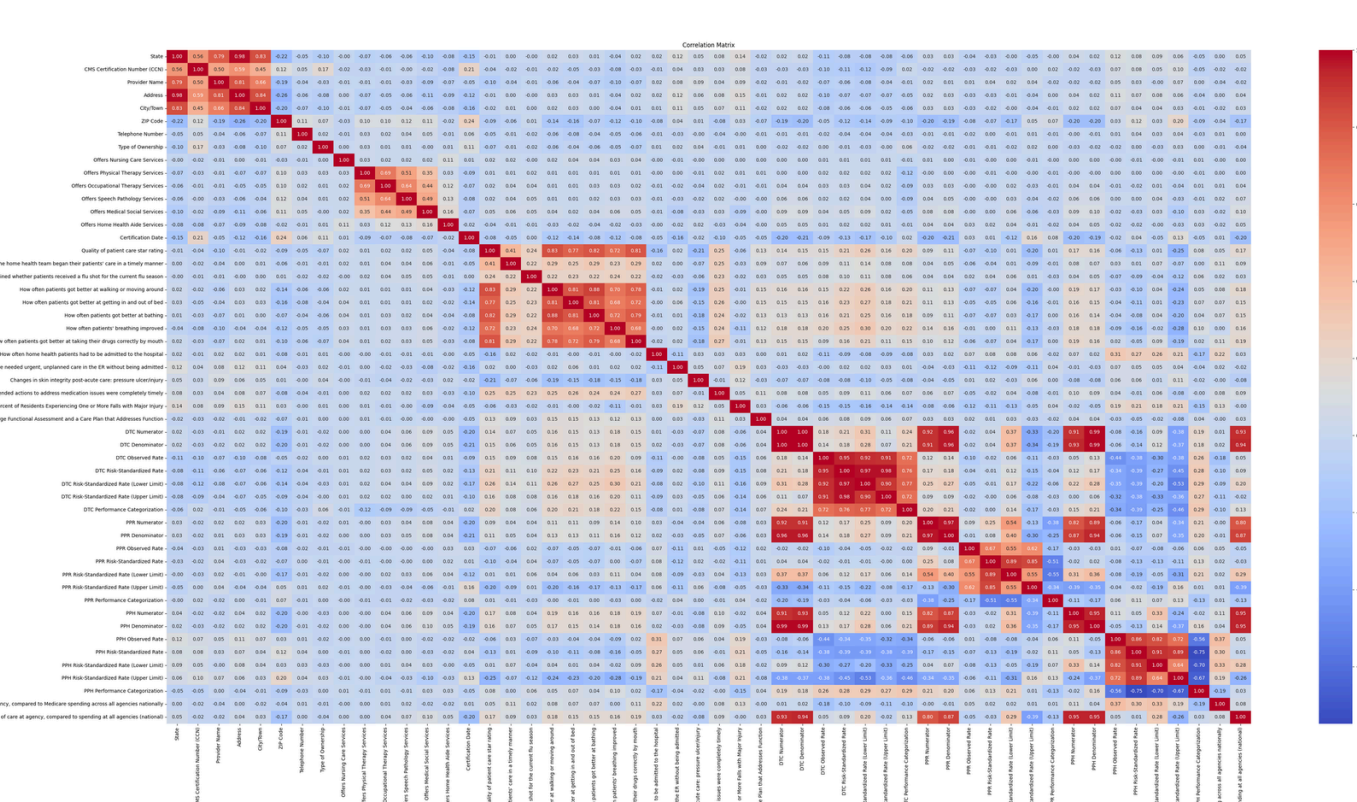
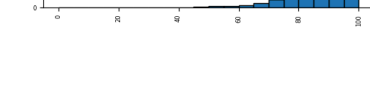
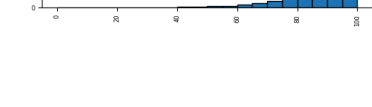
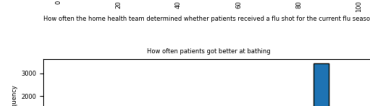
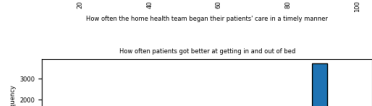
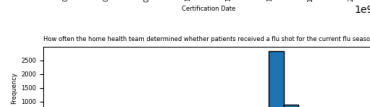
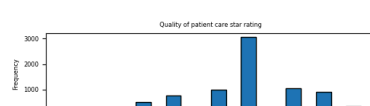
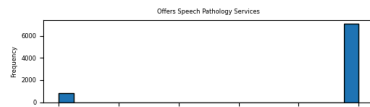
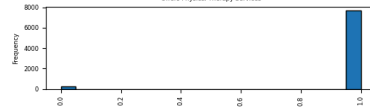
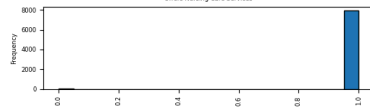
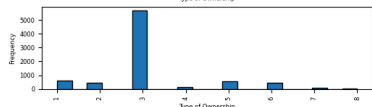
Așadar, după toate transformările am redus setul de date de la (11740, 70) la (8138, 16). Am notat cu (x,y) forma setului de date: x linii, y coloane.

Exploitation analysis

Am realizat o analiză exploratorie a setului de date, unde pentru fiecare coloană am analizat valorile medii, mediana și am afișat datele sub forma unor histograme și a unui heatmap.

```
count    7938.000000
mean      12.228294
```

```
std      3.429199
min      0.000000
25%     10.700000
50%     12.100000
75%     13.500000
max      34.599998
Name: How often patients receiving home health care needed urgent,
unplanned care in the ER without being admitted, dtype: float64
```



Clasificatorii utilizati

a. Rețele neuronale

Am utilizat aceeași arhitectură a rețelei neuronale pentru a obține două modele diferite. Primul model a fost obținut prin antrenarea acesteia cu setul de date inițial, prelucrat conform descrierilor anterioare și utilizarea PCA obținut. Al doilea model a fost obținut prin antrenarea acesteia cu un set de date mai complex, obținut prin concatenarea setului de date initial cu cel găsit la adresa <https://data.cms.gov/provider-data/dataset/ccn4-8vby>, acesta având coloane comune cu setul de date initial și, cu ajutorul modulului pandas, am reușit să concatenăm după coloanele comune. Desigur, și peste setul de date obținut prin concatenare am performat diverse operații: ștergerea liniilor cu valori insuficiente, înlocuiri ale unor valori nule, performarea PCA. De asemenea, unele clase erau subreprezentate și de aceea modelul inițial nu a făcut preziceri bune, deoarece pe unele clase nu se antrena destul. Pentru a rezolva acest neajuns, am augmentat data setul cu alte instanțe.

Arhitectura rețelei

Strat de intrare: Numărul de neuroni este determinat de dimensiunea datelor de intrare. Având în vedere că am folosit aceeași arhitectura pentru două dataseturi, numărul de neuroni în stratul de input este transmis dinamic, în constructor.

Două straturi ascunse: Am utilizat două straturi ascunse, unul de dimensiune 128, iar celălalt de dimensiune 64 care folosesc funcția de activare ReLU.

Strat de ieșire: Folosește funcția de activare Softmax pentru clasificare multi-clasă. Acest strat are 5 neuroni, câte unul pentru

fiecare clasa. Reprezentarea claselor am ales-o conform unor codificari ale valorilor continue din dataset, astfel:

```
bins = [1.0, 1.5, 2.5, 3.5, 4.5, 5.2]
labels = [0, 1, 2, 3, 4]
```

Funcția de cost: Pentru a evalua performanța rețelei, se utilizează funcția de cost **Cross-Entropy**. Aceasta este o alegere comună pentru problemele de clasificare multi-clasă.

Algoritmul de optimizare: Rețeaua este antrenată folosind algoritmul de optimizare **Adam**. Adam este o variantă a gradient descent care adaugă moment și adaptivitate pentru a ajuta la convergența mai rapidă a modelului.

Antrenarea și evaluarea: Datele de antrenare sunt alimentate în rețea în mini-batchuri, iar parametrii rețelei sunt actualizați iterativ folosind algoritmul Adam și funcția de cost Cross-Entropy pentru a minimiza eroarea. Pe măsură ce modelul este antrenat, este important să monitorizezi performanța acestuia pe un set de date de validare pentru a evita supraînvățarea.

După antrenament, rețeaua este testată pe un set de date separat pentru a evalua performanța sa finală.

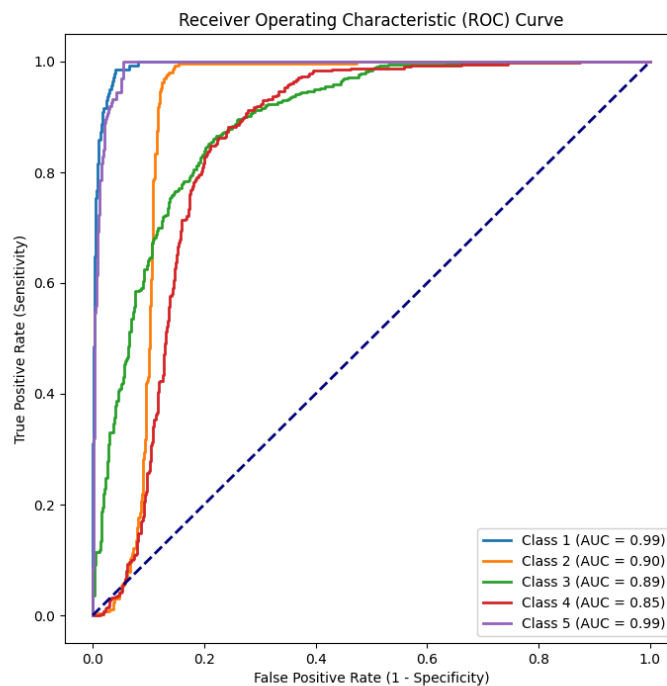
Evaluarea

Pentru setul de date inițial, având dimensiunea de (8138, 16), am obținut o acuratețe de **97%** pe setul de testare. Alte metrice ce descriu performanța sunt:

```
Accuracy is: 0.971983288277218
For 0
    Precision: 0.9730458221024259
    Recall: 0.9691275167785235
    F1 score: 0.9710827168796233
For 1
    Precision: 0.9640122511485452
    Recall: 0.9699537750385208
    F1 score: 0.9669738863287249
For 2
    Precision: 0.9778830963665087
    Recall: 0.9766487851057116
```

```
F1 score: 0.9772655509946322
For 3
Precision: 0.9689169499757163
Recall: 0.9628378378378378
F1 score: 0.9658678286129266
For 4
Precision: 0.9688221709006929
Recall: 0.9824355971896955
F1 score: 0.9755813953488373
```

Analiza ROC a antrenarii pe acest dataset arată după cum urmează:



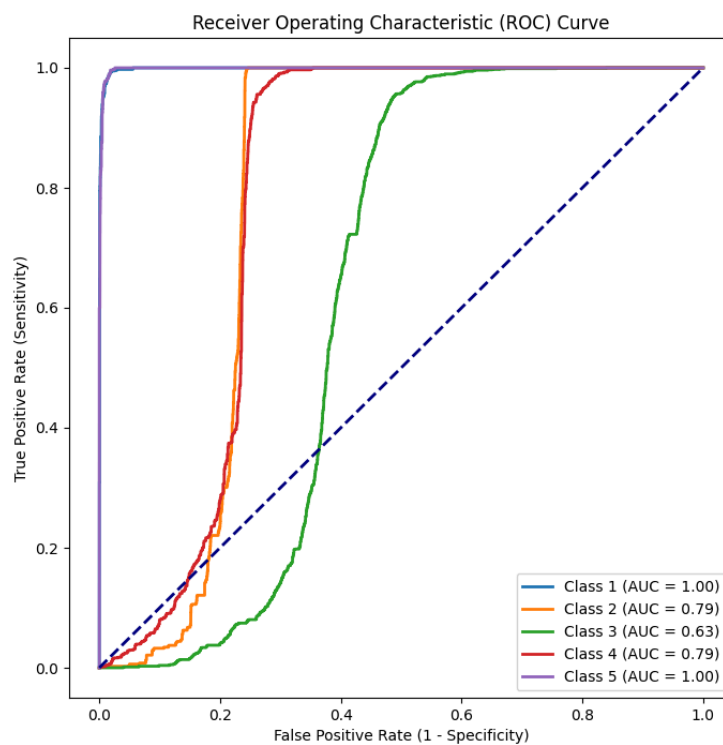
Pentru datasetul concatenat si augmentat, avand dimensiunea de (40841, 18), am obtinut o acuratețe de **98-99%** pe setul de testare. De asemenea, acestea sunt si alte metrice de performanta care descriu analiza rețelei:


```

Test data
Accuracy is: 0.9905741216795202
For 0
    Precision: 0.9987179487179487
    Recall: 1.0
    F1 score: 0.9993585631815266
For 1
    Precision: 0.9922434367541766
    Recall: 0.9993990384615384
    F1 score: 0.9958083832335329
For 2
    Precision: 0.9986486486486487
    Recall: 0.9560155239327296
    F1 score: 0.976867151354924
For 3
    Precision: 0.9717784877529286
    Recall: 0.9956355701036552
    F1 score: 0.9835623821072488
For 4
    Precision: 0.9955555555555555
    Recall: 1.0
    F1 score: 0.9977728285077951

```

Analiza ROC este reprezentată în figura următoare:



Observam ca acuratețea pe al doilea set de date este mai buna, acest fapt fiind datorat faptului ca al doilea set de date cuprinde mai multe instanțe și, de asemenea, prin tehnica augmentarii, am reușit sa oferim modelului cat mai multe exemple pentru fiecare clasa pe care sa se antreneze.

Pentru următorii algoritmi de clasificare am folosit biblioteca sklearn.

b. Support Vector Machines

Principiul de bază al SVM constă în găsirea unui hiperplan optim care poate separa eficient două clase într-un spațiu multidimensional. Acest hiperplan este ales în așa fel încât să maximizeze distanța (marginea) dintre punctele cele mai apropiate ale claselor, numite vectori de suport.

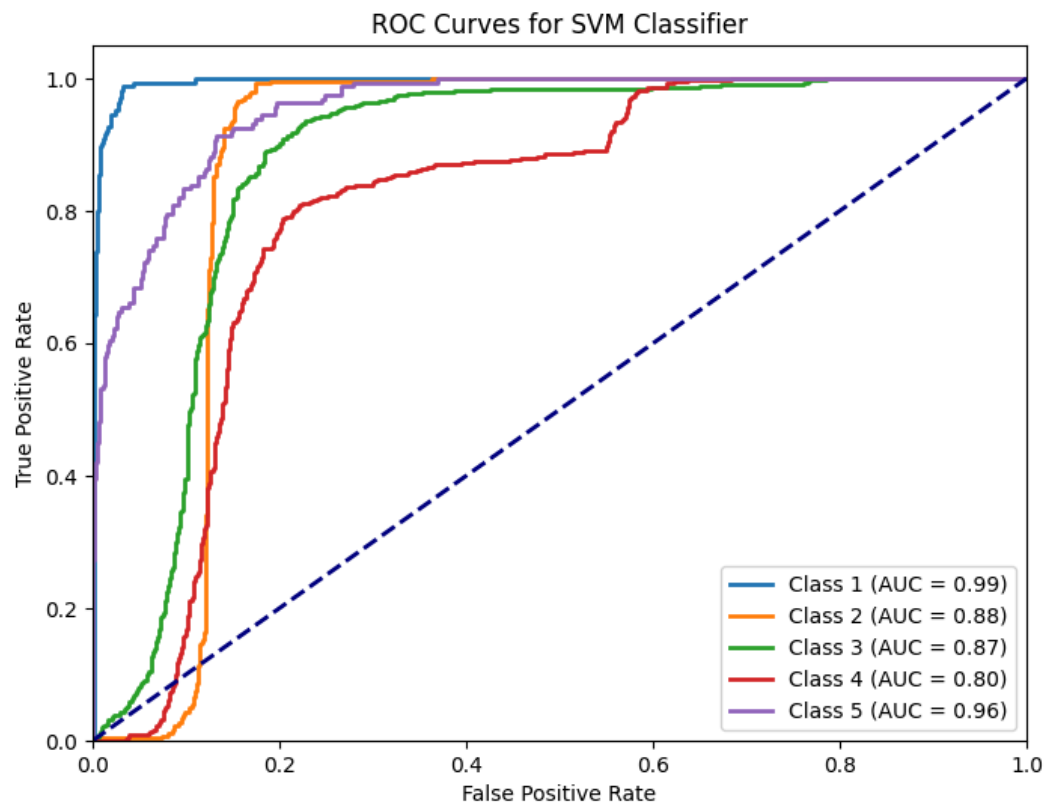
De asemenea, experimentand si observand ca unele instanțe sunt clasificate greșit, am decis sa adaugam o pondere mai mare pentru instanțele din clasa 2 pentru a ajuta algoritmul de clasificare să se “concentreze” mai multe pe ele.

Rezultatul pentru ponderile neschimbate este următorul:
(AUC = Area Under the Curve)

```
custom_weights = {1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0, 5: 1.0}
```

Train Accuracy - : 0.88264

Test Accuracy (SVC) - : 0.89066



Classification Report (SVC):

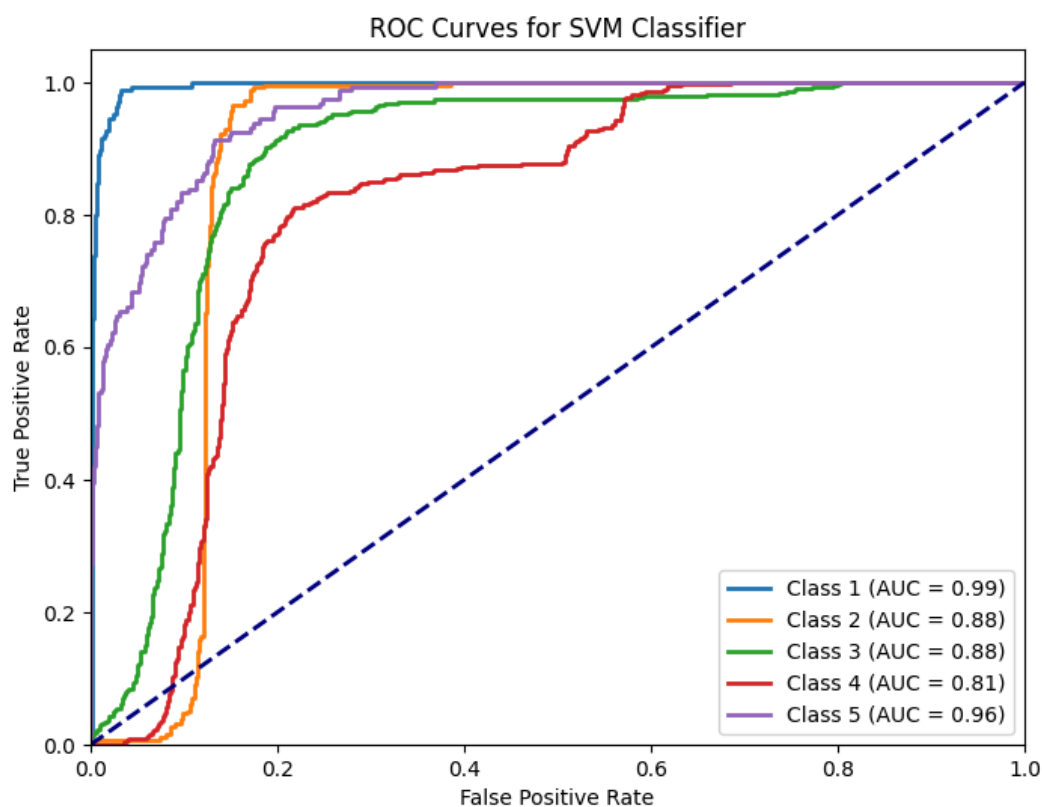
	precision	recall	f1-score	support
1	0.90	0.88	0.89	165
2	0.87	0.91	0.89	256
3	0.94	0.93	0.93	618
4	0.86	0.86	0.86	427
5	0.81	0.81	0.81	162
accuracy			0.89	1628
macro avg	0.88	0.88	0.88	1628
weighted avg	0.89	0.89	0.89	1628

Iar rezultatul pentru ponderile schimbate:

```
custom_weights = {1: 1.0, 2: 1.5, 3: 1.0, 4: 1.0, 5: 1.0}
```

Train Accuracy - : 0.88295

Test Accuracy (SVC) - : 0.89251



Classification Report (SVC):

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

1	0.92	0.86	0.89	165
2	0.86	0.95	0.90	256
3	0.94	0.92	0.93	618
4	0.86	0.86	0.86	427
5	0.81	0.81	0.81	162

accuracy			0.89	1628
macro avg	0.88	0.88	0.88	1628
weighted avg	0.89	0.89	0.89	1628

După cum ne așteptam, se vede o îmbunătățire în metricile de performanță ale algoritmului pentru clasa 2. O comparație frumoasă între prima versiune a clasificatorilor (neoptimizați) și cea curentă (optimizați + augmentarea setului de date) se poate vedea la [Added new ROC Screenshots for all classifiers. · CurcudelTeodor/FeedbackHHC@0cdaef7 \(github.com\)](https://github.com/CurcudelTeodor/FeedbackHHC@0cdaef7)

Avem și o funcționalitate pentru a prezice o nouă instanță dată de utilizator cu ajutorul funcției `predict_with_classifier`.

```
def predict_with_classifier(model, scaler, instance):
    instance_list = list(csv.reader(StringIO(instance)))[0]
    instance_list[1] = int(instance_list[1])
    instance_list[5] = int(instance_list[5])

    # include instance in the data frame
    df = handlers.include_instance(instance_list)

    # extract the PCA-transformed instance
    instance_pca = df.iloc[-1]

    # standardize features using the provided scaler
    instance_array = scaler.transform(instance_pca.values.reshape(1, -1))

    # make prediction using the Random Forest model
    prediction = model.predict(instance_array)

    return prediction
```

Pentru conveniența și readability, am implementat câte un handler în fiecare din clasificatori pentru a prezice cum va fi clasificată instanța de către clasificatorul respectiv

```
def handle_predict_svm(instance, svm_model, scaler):  
    prediction = predict_with_classifier(svm_model, scaler, instance)  
    return {'svm_prediction': prediction}
```

Predicția pentru instanța data de cod este:

```
{'svm_prediction': array([5], dtype=int64)}
```

ceea ce corespunde cu eticheta reală a instanței.

```
{'mlp_prediction': array([5], dtype=int64)}
```

c. Multi-layer perceptron

Multi-layer Perceptron (MLP) este o formă de rețea neuronală învățată utilizând metode de învățare supervizată. Chiar dacă avem doar câte 5, 7 și respectiv 6 neuroni pe cele 3 straturi ascunse, acuratețea este de **89%**, o valoare foarte buna avand in vedere setul mare de date și neuronii relativ puțini.

```
mlp_classifier = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 7, 6),  
max_iter=1000, random_state=42, alpha=0.1)
```

Funcția de activare este tot ReLu așa cum am făcut și la rețeaua neuronală cu **98%-99%** acuratețe.

Metricile de performanta sunt următoarele:

Accuracy (MLP) on training set - : 0.88264

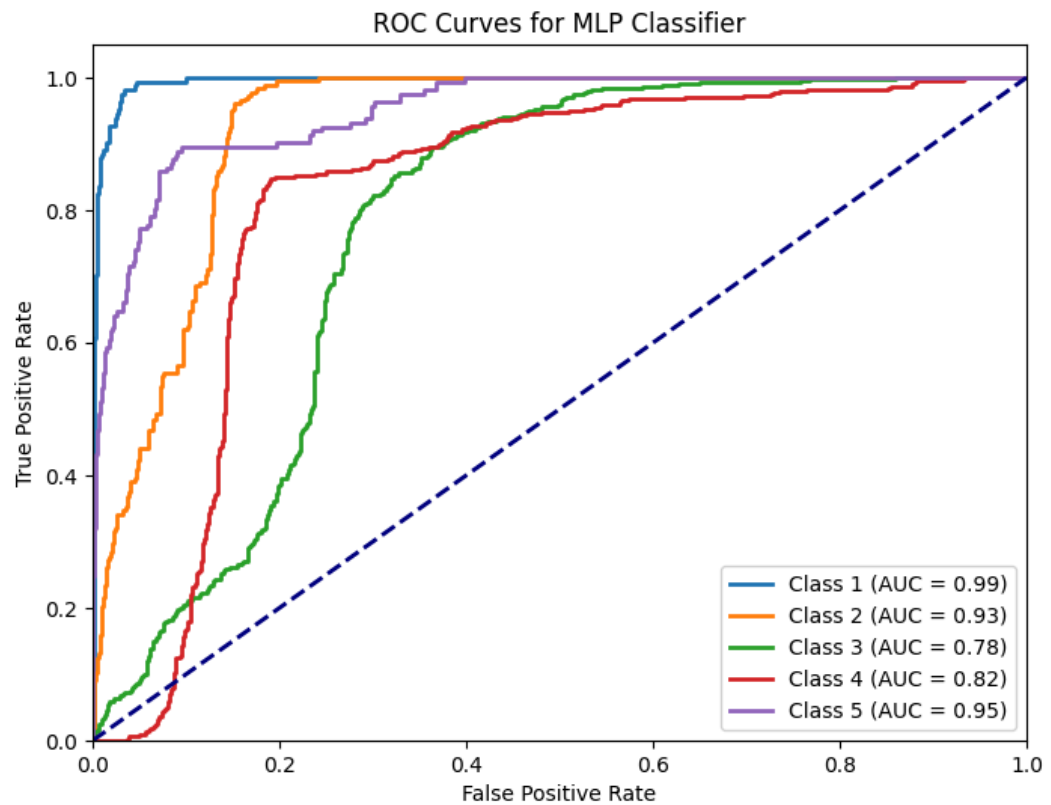
Accuracy (MLP) on test set - : 0.88943

Classification Report (MLP):

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

1	0.89	0.90	0.89	165
2	0.88	0.90	0.89	256
3	0.94	0.92	0.93	618
4	0.85	0.86	0.86	427
5	0.82	0.81	0.82	162

accuracy				0.89	1628
macro avg	0.88	0.88	0.88	0.88	1628
weighted avg	0.89	0.89	0.89	0.89	1628



Necesitatea introducerii acestui clasificator mai simplu, mai lightweight (mai puțini neuroni pe straturile hidden) o vom vedea în continuare.

Dacă dorim să facem o predicție pentru o instanță nouă, putem proceda în mod similar cu ce am descris mai sus la SVM.

Acesta este output-ul dacă vrem să prezicem aceeași instanță dată ca exemplu la SVM. Vom observa că este precisă clasa 5.

```
{'mlp_prediction': array([5], dtype=int64)}
```

Deci necesitatea este următoarea: Putem prezice corect eticheta unei instanțe chiar și cu o rețea neuronală mai lightweight, însă pentru predicții cât mai corecte, mai aproape de adevăr utilizăm rețeaua de la punctul a) cu **98%-99%** acuratețe la testare.

Remarcăm faptul că predicția este tot clasa 5 indicând o acuratețe bună. Putem fi mai siguri că aceasta este clasa reală a instanței dacă 2 clasificatori cu acuratețe mare la testare (**89%-90%**) au clasificat-o în același fel.

d. Random Forest

Este un algoritm de tip ansamblist pentru că formează mai mulți arbori de decizie mai mici pe care apoi îi combină pentru a avea o acuratețe mai mare astfel obținând și o predicție mai robustă și precisă.

Pentru a stabili valorile optime pentru parametrii și hiperparametrii clasificatorului am implementat un Grid Search ceea ce înseamnă că am creat diferiți clasificatori de tip Random Forest cu configurații diferite apoi i-am antrenat. La sfârșit am considerat combinația de valori pentru care s-au obținut cele mai bune

metrici. Am observat ca ceea ce influenteaza cel mai mult acuratea a fost numarul de estimatori (arbori mai mici). Cea mai buna valoare (sweet spot) pentru acesta a fost de 70.

```
def grid_search_random_forest(X_train, y_train):  
    # define the parameter grid  
    param_grid = {  
        'n_estimators': [50, 70, 100],  
        'max_depth': [None, 10, 20, 30],  
        'min_samples_split': [2, 5, 10],  
        'min_samples_leaf': [1, 2, 4],  
        'criterion': ['gini', 'entropy']  
    }  
    . . .
```

Și aici am folosit ponderi (weights) pentru a clasifica corect instanțele cu clasa 2. Rezultatele de data aceasta se schimba dramatic.

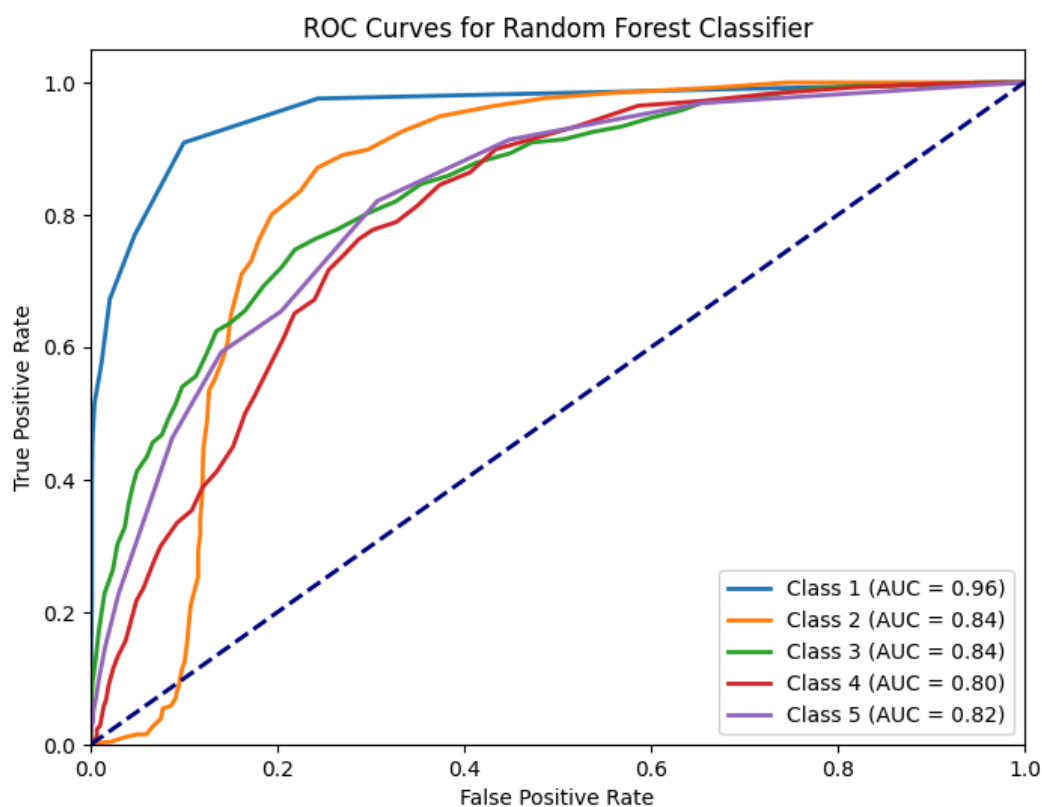
```
class_weights = {1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0, 5: 1.0}
```

Train Accuracy - : 1.00000

Test Accuracy - : 0.81020

Classification Report:

	precision	recall	f1-score	support
1	0.95	0.82	0.88	165
2	0.80	0.81	0.80	256
3	0.80	0.82	0.81	618
4	0.73	0.79	0.76	427
5	0.98	0.83	0.90	162
accuracy				0.81 1628
macro avg	0.85	0.81	0.83	1628
weighted avg	0.82	0.81	0.81	1628



Metricile de performanta obținute sunt mai jos cu ponderea schimbată pentru clasa 2:

```
class_weights = {1: 1.0, 2: 3.0, 3: 1.0, 4: 1.0, 5: 1.0}
```

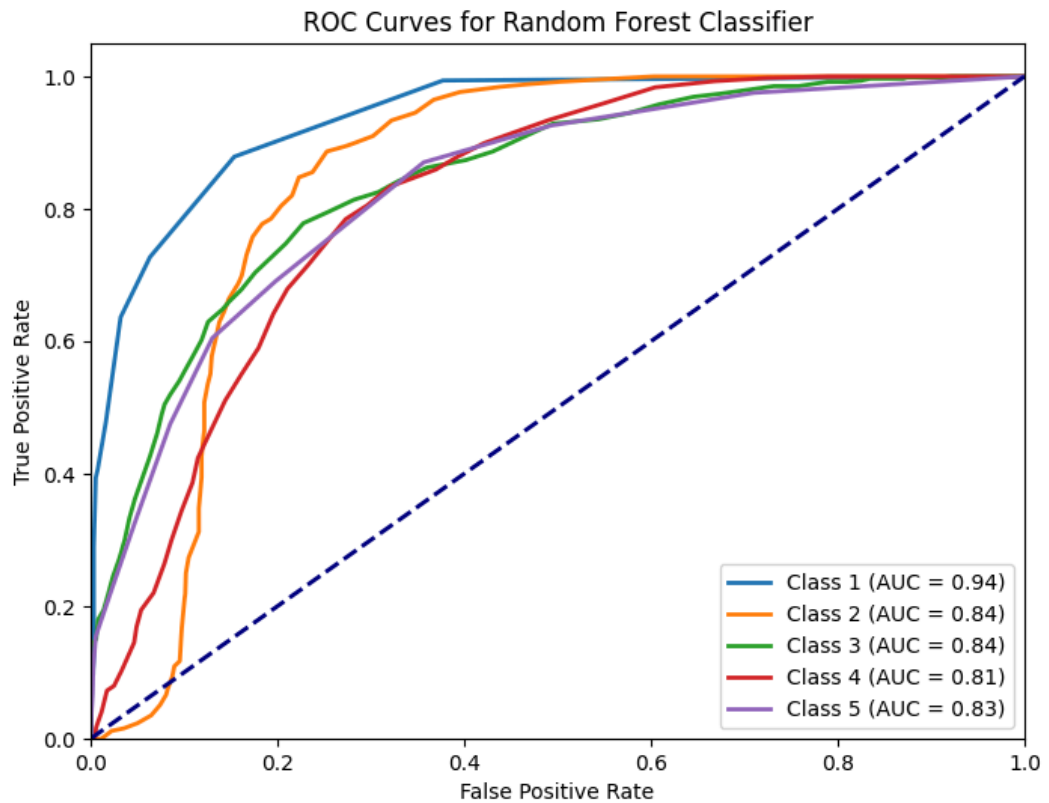
Train Accuracy - : 1.00000

Test Accuracy - : 0.84310

Classification Report:

	precision	recall	f1-score	support
1	0.94	0.81	0.87	165
2	0.81	0.81	0.81	256
3	0.80	0.85	0.83	618
4	0.77	0.79	0.78	427
5	0.96	0.85	0.90	162

accuracy			0.82	1628
macro avg	0.86	0.82	0.84	1628
weighted avg	0.83	0.82	0.82	1628



Și aici avem implementată funcționalitatea pentru a prezice o instanță nouă, însă din cauză că acuratețea este ceva mai mică, clasa prezisă este 3 pentru aceeași instanță de la ceilalți 2 clasificatori.

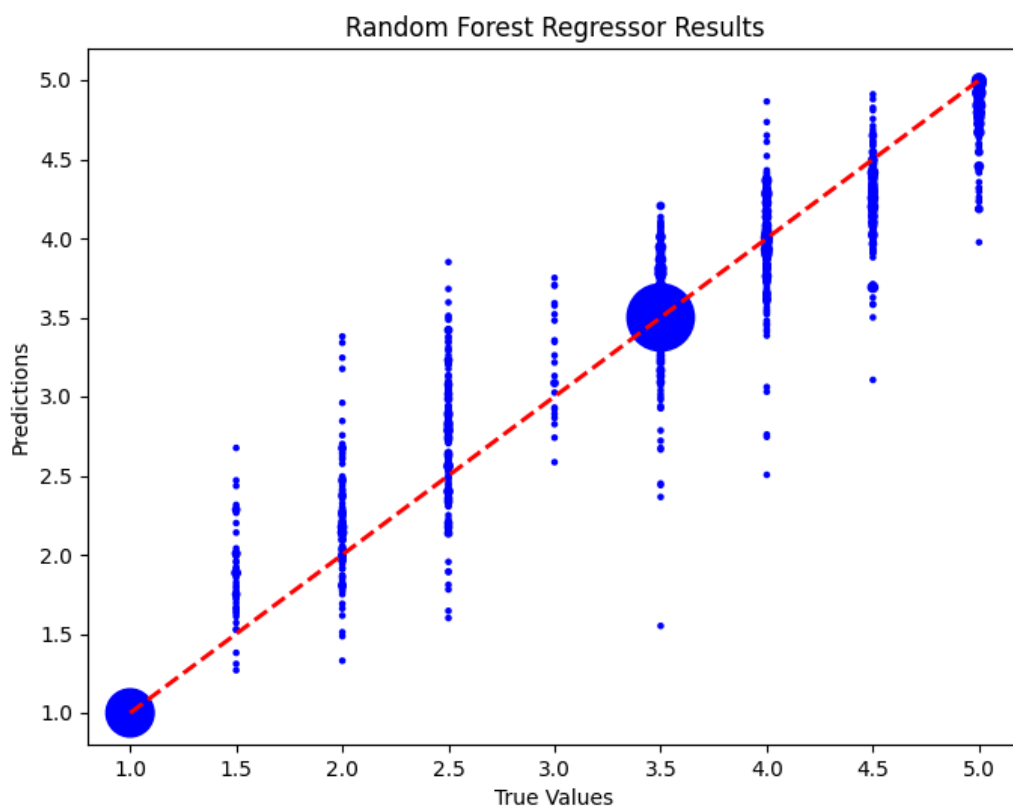
```
{'random_forest_prediction': array([3], dtype=int64)}
```

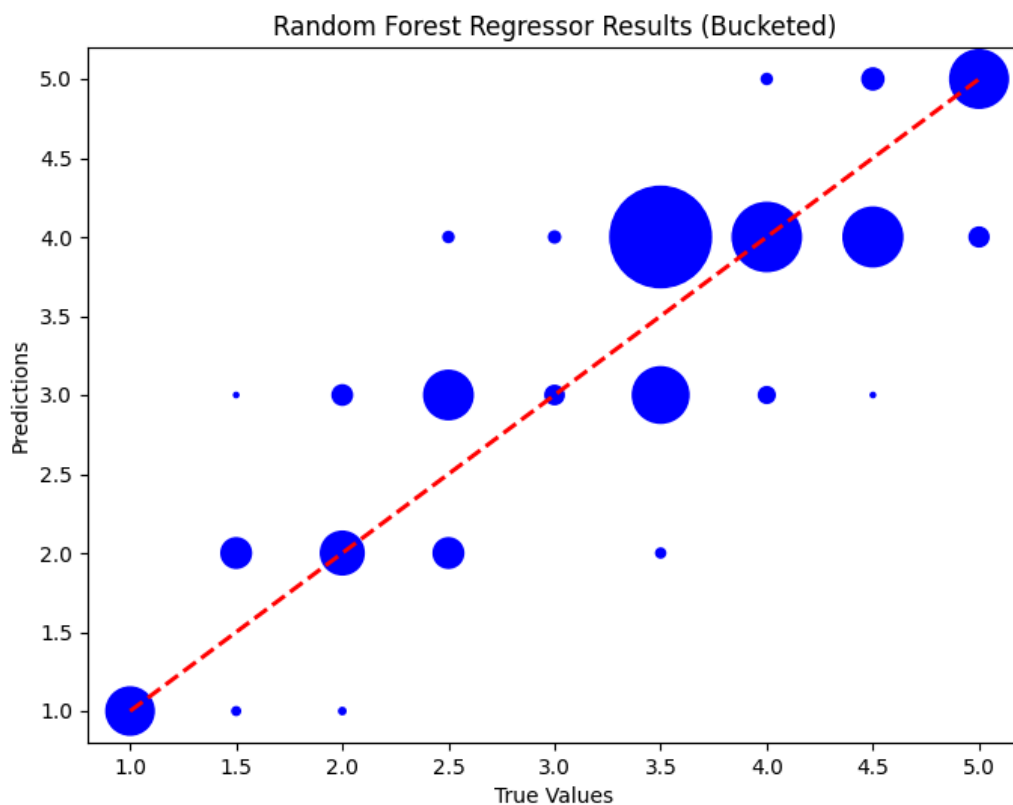
e. Random Forest Regressor

În această versiune a clasificatorului considerăm clasele ca fiind valori continue și nu valori discrete ca până acum. Din această cauză, antrenarea va dura considerabil mai mult decât în cazul celorlalți clasificatori.

Metricile și graficele de performanță sunt mai jos:

Mean Squared Error: 0.09823925061425062





Observam ca de cele mai multe ori, valorile prezise sunt apropiate de cele reale.

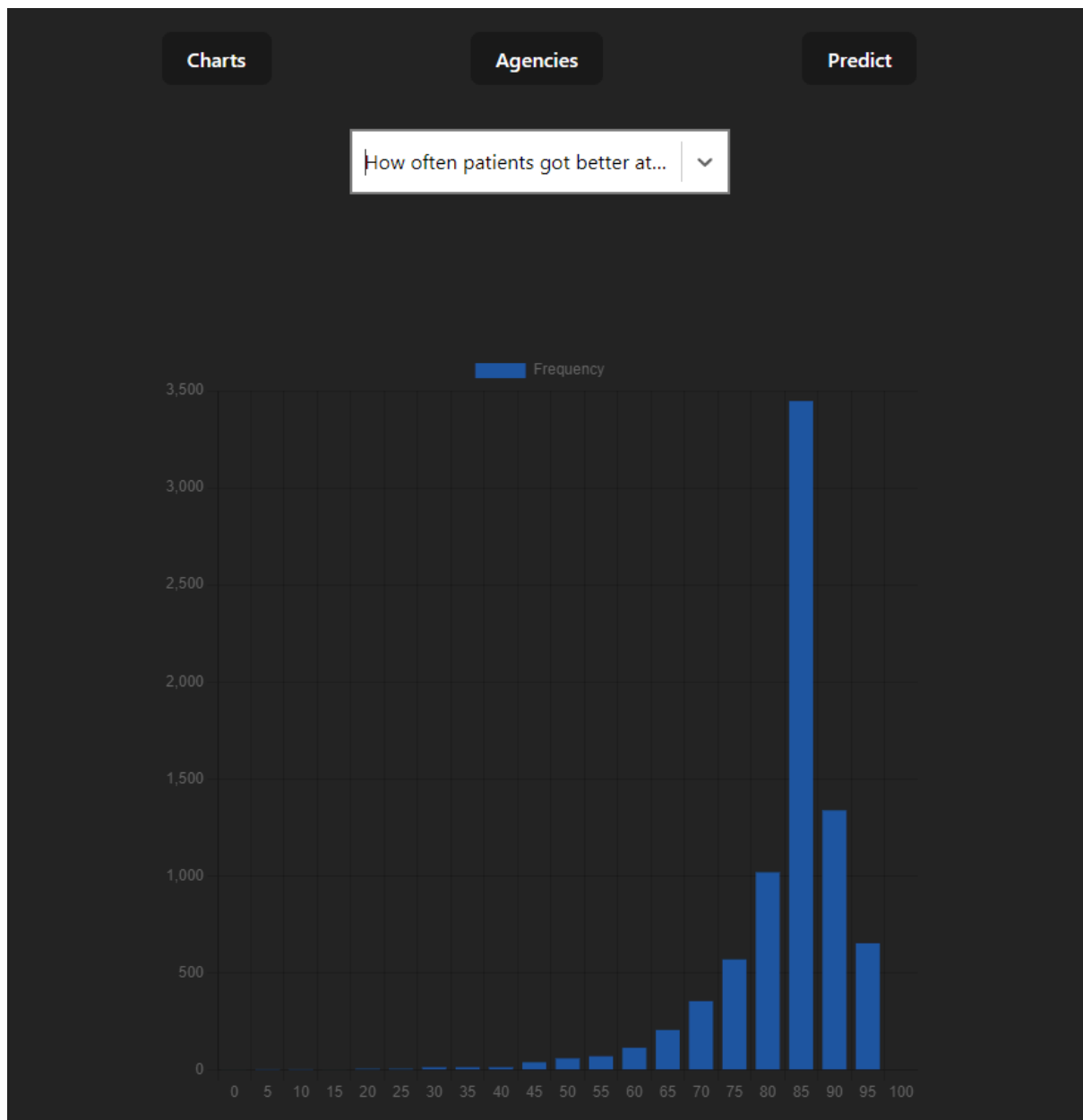
Predicția pentru aceeași instanța de la ceilalți clasificatori este de

```
{'random_forest_regressor_prediction': array([2.945])}
```

Observam ca este o valoare reala, nu intreaga!!

Interfață Web:

Pentru ca datele să fie mai ușor de vizualizat, am construit și o pagină web. În cadrul ei putem selecta coloanele specifice din setul de date și vizualiza histograme:



O altă opțiune disponibilă este cea de a căuta date despre o anumită agenție medicală. Se introduce numele agenției și codul poștal,

În timp ce scrii apar mai multe opțiuni asemănătoare pentru a simplifica căutarea.

Charts

Agencies

Predict

PROVIDENCE HOME HEAL

99508

Search


Home health agency

PROVIDENCE HOME HEALTH ALASKA

OFFICE LOCATION
4001 DALE STREET, SUITE 101
ANCHORAGE, AK 99508

PHONE NUMBER
9075630130

Quality of patient care star rating



Services Provided

Offers Home Health Aide Services

✓ Yes

Offers Medical Social Services

✓ Yes

Offers Nursing Care Services

✓ Yes

Offers Occupational Therapy Services

✓ Yes

Offers Physical Therapy Services

✓ Yes

Offers Speech Pathology Services

✓ Yes

De asemenea putem introduce o instanță pe care vrem ca algoritmul nostru sa o clasifice:

Charts

Agencies

Predict

AK,027020,"1ST CHOICE H

Go!

Classifier	Prediction
nn_base_prediction	2

Componenta echipei:

- Curcudel Teodor
- Duluță George
- Harton Amalia
- Olariu Andreea
- Turtureanu Cosmin