UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ROMÂNĂ

LUCRARE DE LICENŢĂ

Diagnosticarea bolilor cardiovasculare prin tehnici de învățare automată

Conducători ştiinţifici

Drd. Daniel Boța
Conf. dr. Adrian Sterca

Absolvent
Talpoș Darius

2024

ABSTRACT

This paper is meant to show the research and its results regarding developing a classification model for cardiac diseases. Several machine learning algorithms and transformation methods have been tested in order to obtain the final result. The first chapter is an introductory one, presenting the problem at hand and the reason for searching for a solution. The second chapter starts with an introduction to basic concepts of machine learning, followed by presenting multiple algorithms chosen to be evaluated for this problem. The next chapter focuses on what the classification problem implies, as well as on how the data used for it is evaluated and transformed in order to improve the result. The fourth chapter focuses on the steps taken towards developing the model, as well as to how the website, the related application, was built. The ending reflects on the results achieved and points out steps towards the betterment of both model and application that could be taken, should either of them be going forward in development.

Cuprins

Capitolul 1

Introducere

Bolile cardiace, cunoscute şi drept boli cardiovasculare, au risc de apariție la toate categoriile de vârstă, motiv pentru care monitorizarea acestora este esențială. Există factori care pot să crească sau să scadă riscul dezvoltării acestor afecțiuni. Alimentația nesănătoasă, hipertensiunea arterială, sedentarismul, consumul excesiv de alcool și nivelul ridicat de glucoză/colesterol sunt doar câțiva factori care pot să determine apariția acestor boli [Mara].

Informatica a avansat considerabil în ultimii ani, în special în ceea ce priveşte domeniul inteligenţei artificiale. Învățarea automată, o subcategorie a inteligenței artificiale, ne permite să obținem modele care cu o anumită șansă de corectitudine, au capacitatea de a evalua datele primite. Învățarea supervizată, o categorie de învățare automată, este utilizată în rezolvarea problemelor de clasificare, unde pe baza trăsăturilor sale, o instanță este încadrată într-o categorie.

Scopul acestei lucrări este să îmbine aceste două domenii, astfel încât să se poată determina prin intermediul anumitor trăsături referitoare la sănătatea individului dacă persoana suferă sau nu de o boală cardiovasculară.

1.1 Motivație

Conform Organizației Mondiale a Sănătății, bolile cardiovasculare reprezintă principalul motiv al deceselor la nivel global. În anul 2019, acestea au cauzat 32% din decese, dintre care 85% au fost atacuri de cord sau accidente vasculare cerebrale [Org].

Potrivit Societății Române de Cardiologie, bolile cardiace sunt cauza decesului a 60% dintre români. Până la o anumită vârstă, afecțiunile cardiovasculare au un risc mai mare de apariție în cazul bărbaților. După menopauză, riscul este la fel de ridicat pentru ambele sexe [Marb].

Motivul pentru care am realizat acestă aplicație este dorința de a putea avea un

mod accesibil de monitorizare al sănătății. Cauzele apariției unei boli cardiovasculare sunt diverse, așadar detectarea la timp a problemelor de sănătate este crucială. Dacă afecțiunea este descoperită în timp util, pot fi stopate complicațiile și astfel crește șansa de supraviețuire a individului. Toate datele care vor fi cerute pot fi obținute fără a realiza un control medical, dat fiind faptul că există aparate pentru verificarea tensiunii, glicemiei și nivelului colesterolului disponibile în comerț.

Chiar dacă rezultatul poate fi unul negativ, persoana în cauză poate conştientiza astfel existenţa unor obiceiuri nesănătoase sau a unor valori crescute și să trateze ca atare acești posibili factori de risc.

1.2 Obiective

Prin utilizarea unui set de date referitor la bolile cardiovasculare, această lucrare urmărește testarea mai multor algoritmi de clasificare și metode de transformare a datelor astfel încât să se obțină cel mai performant model. Acesta va permite cu o anumită acuratețe să se deducă pe baza datelor dacă persoana suferă sau nu de o boală cardiovasculară. Modelul va fi ulterior integrat într-o aplicație, un site web.

Aplicația web rezultată va fi realizată astfel încăt să poată fi utilizată rapid şi eficient. Ea va fi un portal care să permită unui utilizator să își introducă datele necesare şi să primească pe baza acestora un rezultat, negativ sau pozitiv, referitor la existența unei boli cardiovasculare.

Scopul acestui studiu nu este de a înlocui un control specializat realizat de o persoană cu expertiză medicală, dat fiind că rezultatul nu va fi în mod garantat corect. În schimb, aplicația și modelul au rolul de a oferi o metodă alternativă de verificare, o a doua părere, în ceea ce privește starea de sănătate a sistemului cardiovascular.

1.3 Structură

Această lucrare este structurată pe 4 capitole și conține 21 referințe bibliografice: articole, cărți, site-uri web de specialitate, documentații pentru limbaje de programare/librării/framework-uri.

Primul capitol prezintă noţiuni de bază referitoare la învățarea automată (ce este aceasta, care sunt diferitele tipuri de învățare automată), și descrie mai mulți algoritmi de învățare supervizată (Regresie Logistică, k-Nearest Neighbors etc.) și algoritmi de tip ansamblu (AdaBoost Classifier, Random Forest etc.).

Cel de-al doilea capitol descrie clasificarea ca metodă de învățare automată supervizată și cum se diferențiază de regresie, diferitele modalități de transformare a datelor din set, dar și metricile utilizate pentru a analiza performanța modelului

realizat.

Capitolul trei prezintă modul în care au fost realizate modelul și aplicația web. Sunt prezentate scopul și funcționalitățile aplicației *HeartVitality* , componentele client și server ale acesteia, structurarea sa și un manual de utilizare. Ultimul subcapitol descrie setul de date utilizat și modul în care au fost comparate rezultatele diferiților algoritmi spre obținerea celui mai eficient rezultat.

Ultimul capitol se va concentra asupra rezultatelor obținute în ceea ce privește modelul și aplicația Web. De asemenea, vor fi prezentate sugestii pentru îmbunătățirea eficienței modelului, dar și noi funcționalități care ar fi benefice site-ului.

1.4 Utilizarea instrumentelor IA generative

Autorul nu a folosit instrumente IA generative în elaborarea acestei lucrări.

Capitolul 2

Învățarea automată

Arthur Samuel a definit în anul 1959 învățarea automată (en. ,,machine learning") ca fiind ,,domeniul de studiu care permite calculatoarelor să înveţe fără a fi explicit programate" [Sam59]. Cu alte cuvinte, învățarea automată reprezintă implementarea de algoritmi care să fie capabili să învețe din datele furnizate la intrare fără a primi instrucțiuni suplimentare. Modul prin care algoritmii învață pe baza datelor presupune recunoașterea unor tipare astfel încât rezultatul să fie cât mai apropiat de realitate [GS13]. Există două tipuri principale de învățare automată: învățare supervizată (en. ,,supervised learning") și învățare nesupervizată (en. ,,unsupervised learning").

Învățarea supervizată presupune faptul că instanțele setului de date au o etichetă (en. ,,label"). Eticheta poate reprezenta o clasă sau o valoare numerică, iar scopul algoritmilor de învățare supervizată este să obțină eticheta corespunzătoare pentru noile instanțe. Învățarea supervizată încearcă astfel să rezolve probleme de clasificare și de regresie. Învățarea nesupervizată diferă de cea supervizată prin faptul că instanțele nu vor avea o etichetă. Algoritmii de învățare nesupervizată încearcă să rezolve problemele de clustering. Problema prezentată în această lucrare este una de clasificare, astfel algoritmii utilizați vor fi aleși în mod corespunzător. Conform [Bro16], rezolvarea unei probleme de învățăre automată presupune mai mulți pași:

1. Definirea problemei (înțelegerea contextului problemei și a modului de abordare necesar);
2. Analiza datelor (înțelegerea formatului datelor și a atributelor lor);
3. Pregătirea datelor (aplicarea de transformări asupra datelor în scopul îmbunătățirii rezultatelor);
4. Evaluarea algoritmilor (testarea mai multor algoritmi și compararea rezultatelor lor);

5. Îmbunătățirea rezultatelor (testarea algoritmilor precedent eficienți după modificarea anumitor parametri);

6. Prezentarea rezultatelor (finalizarea și salvarea modelului).

2.1 Algoritmi de învățare supervizată

Învățarea supervizată este un proces prin intermediul căruia, utilizând un algoritm de învățare automată, se obține un model, care cu o anumită acurateţe, poate estima clasa unei instanțe de date necunoscute. Algoritmii de învățare supervizată utilizați în această lucrare sunt:

- Regresie logistică;
- Linear Discriminant Analysis;
- k-Nearest Neighbors;
- Arbori de decizie;
- Naïve Bayes;
- Support Vector Machines.

2.1.1 Regresie logistică

Regresia logistică (en. ,,Logistic Regression") este un model statistic utilizat în analiza predictivă și de clasificare. Aceasta se folosește de funcția logistică, o funcție matematică a cărei valori se află în intervalul $[0, 1]$. Funcția logistică este o particularizare a funcției sigmoid, a cărei principală trăsătură este forma de S a graficului. Formula funcției sigmoid este următoarea [Har19]:

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

Dat fiind intervalul între care funcția poate lua valori, se vor crea două grupe: cea a valorilor de 0, în care vor intra valorile din intervalul $[0, 0.5]$, și cea a valorilor de 1, în care vor intra valorile din intervalul $[0.5, 1]$.

Regresia logistică caută parametri care să se potrivească cel mai bine funcției sigmoid. Ea funcționează cu valori numerice și are drept dezavantaj faptul că este predispusă la underfitting (performanță scăzută din cauza faptului că modelul nu este capabil să analizeze datele în mod corespunzător), dar și posibilitatea de a avea o acuratețe scăzută [Har19].

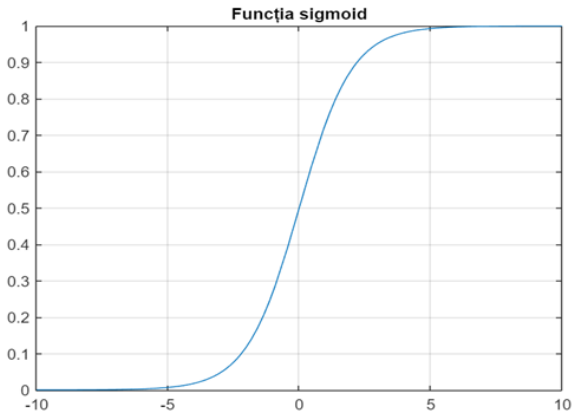

Figura 2.1: Valorile luate de funcția sigmoid pe intervalul [-10,10].

2.1.2 Linear Discriminant Analysis

Linear Discriminant Analysis este un algoritm utilizat pentru analiza de clasificare. Conform [TGIH17], LDA presupune că datele au o distribuție Gaussiană și încearcă să transforme atributele instanțelor într-un spațiu dimensional inferior care să maximizeze separarea între clase prin maximizarea raportului dintre varianța „within-class" și varianța „between-class". Algoritmul urmează trei pași:

- Calcularea variației „between-class" (distanța dintre mediile diferitelor clase);
- Calcularea variației „within-class" (diferența dintre media și instanțele clasei);
- Construirea spațiului dimensional inferior care permite maximizarea variației „between-class" și minimizarea variației „within-class".

2.1.3 k-Nearest Neighbors

k-Nearest Neighbors este un algoritm de clasificare. În etapa de antrenament acesta reține atributele și clasa pentru fiecare instanță, urmând ca pentru fiecare instanță nouă din etapa de testare să compare atributele acesteia cu a instanțelor deja existente. La final, ea va prelua clasa majoritară a celor mai apropiate k instanțe (vecini) din punct de vedere al valorilor acestora [Har19]. Pentru a determina cele mai apropiate instanțe pot fi utilizate mai multe distanțe. Distanța Minkowski, cea utilizată în mod normal, este o generalizare atât a distanței Euclidiene, cât și a celei Manhattan [Bro16].

2.1.4 Arbori de decizie

Decision Tree Learning este un algoritm bazat pe arborii de decizie (en. „decision trees") utilizat în problemele de clasificare și de regresie. Un arbore de decizie este

un graf orientat unde fiecărui nod intern îi corespunde un atribut, ramurilor sale îi corespund valorile sale, iar frunzelor le corespund fiecăreia câte o clasă.

Un dezavantaj al acestui algoritm este faptul că acesta este predispus la overfitting (faptul că modelul a reținut rezultatele datelor de antrenament fără să poată distinge motivul pentru încadrarea lor în acea categorie, astfel fiind incapabil de a încadra corespunzător noile instanțe) [Har19].

2.1.5 Naïve Bayes

Naïve Bayes este un algoritm folosit în problemele de clasificare. Față de alți algoritmi, nu reține care dintre atribute sunt cele mai importante în diferențierea dintre clase. Acest algoritm se bazează pe teorema lui Bayes, enunțată de matematicianul britanic Thomas Bayes, având următoarea formulă:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Astfel, probabilitatea ca evenimentul A să aibă loc, știind că a avut loc evenimentul B, poate fi calculată în funcție de probabilitatea inversă [IBM].

2.1.6 Support Vector Machines

Support Vector Machines sunt modele supervizate cu algoritmi pentru analiza de regresie și de clasificare, fiind în special folositoare atunci când datele au un volum mare. Acest clasificator se bazează pe identificarea unui hiperplan care separă elementele în funcție de clasa căreia îi aparțin, urmând ca noile instanțe să fie clasificate în funcție de poziția avută față de hiperplan.

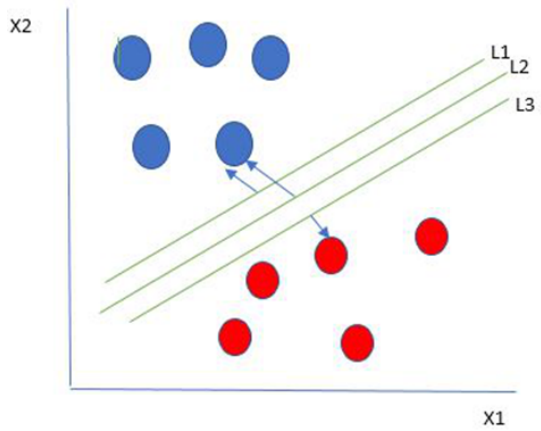

Figura 2.2: Hiperplanuri care delimitează clasele [Gee]

Punctele cele mai apropiate de linia care separă hiperplanurile sunt cunoscute drept vectori de suport (en. ,,support vectors''). Soluția ideală va fi dată de marginea care are distanța maximă posibilă față de vectorii de suport [Bro16].

2.2 Algoritmi de tip ansamblu

Algoritmii de tip ansamblu (cunoscuți și ca meta-algoritmi), presupun combinarea mai multor algoritmi de clasificare. Există mai multe metode de a combina acești algoritmi, două dintre acestea fiind utilizate în realizarea modelului: boosting și bagging.

Boosting-ul presupune antrenarea secvențială a algoritmilor, astfel se încearcă corectarea greșelilor realizate de algoritmii precedenți. Algoritmii de boosting utilizați în cadrul acestei lucrări sunt AdaBoost și Gradient Boosting.

Bagging-ul se referă la crearea mai multor seturi de date de aceeași dimensiune prin alegerea aleatoare a acestora. Asupra fiecărui set format se aplică un algoritm, iar în cadrul noilor instanțe se vor aplica fiecare dintre algoritmi, la final alegându-se rezultatul majoritar. Random Forest și Extra Trees sunt algoritmii de bagging care au fost folosiți în lucrare.

2.2.1 AdaBoost Classifier

AdaBoost (de la ,,Adaptive Boosting'') este un algoritm inventat de Yoav Freund și Robert Schapire. Acesta presupune asignarea unei ponderi pentru fiecare instanță, inițial egală la fiecare. În urma testării, ponderile vor fi modificate astfel încât instanțele incorect clasificare să aibă o pondere mai mare decât celelalte. Algoritmul va încerca astfel să corecteze greșelile din iterația anterioară. Eroarea are formula următoare:

$$\varepsilon = \frac{i}{n}$$

unde i reprezintă numărul de instanțe incorect clasificate, iar n reprezintă numărul total de instanțe. Fiecare clasificator din AdaBoost va avea propria valoare bazată pe eroare, în funcție de care își va ajusta ponderile. Valoarea clasificatorului este calculată în funcție de eroare astfel:

$$\alpha = \frac{1}{2} ln(\frac{1-\varepsilon}{\varepsilon})$$

Având calculată valoarea clasificatorului, vectorul de ponderi poate fi modificat astfel încât instanțele incorect clasificate să aibă o importanță mai mare. Noua pondere pentru instanța i, la iterația $t+1$, se determină astfel:

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{Sum(D)}, \text{ dacă instanța } i \text{ a fost corect clasificată}$$

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{Sum(D)}, \text{ altfel}$$

După actualizarea vectorului, va începe o nouă iterație, iar algoritmul se va finaliza atunci când eroarea obținută va fi 0 sau a fost atins numărul de clasificatori dat de utilizator [Har19].

2.2.2 Gradient Boosting Classifier

Gradient Boosting este un algoritm de boosting. Conceptul de „gradient boosting" a fost inventat de Leo Breiman, iar algoritmul a fost dezvoltat de Jerome H. Friedman.

	Algorithm 1: Gradient_TreeBoost
1	$F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} \Psi\left(y_i, \gamma\right)$
2	For $m = 1$ to M do:
3	$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4	$\{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\tilde{y}_{im}, \mathbf{x}_i\}_1^N)$
5	$\gamma_{lm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{lm}} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \gamma\right)$
6	$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
7	endFor

Figura 2.3: Algoritmul scris de J. Friedman [Fri02]

În figura 2.3 este prezentat algoritmul Gradient Boosting. El presupune antrenarea modelului folosind „reziduurile" (diferențele dintre valorile prezise și cele reale) cu scopul de a optimiza funcția de pierdere $\psi(y, F(x))$ [Fri02]. Funcția de pierdere utilizată poate varia. În cazul acestui program a fost utilizată funcția implicită pentru clasa algoritmului de Gradient Boosting aferentă din biblioteca scikit-learn. Această funcție este pierderea logistică, recomandată pentru problemele de clasificare [sla].

2.2.3 Random Forest Classifier

Random Forest este un algoritm de bagging conceput de Leo Breiman și Adele Cutler. El presupune crearea mai multor arbori de clasificare. Date fiind N instanțe în datele de antrenament, fiecare arbore va alege aleator N instanțe (cu posibilitatea de a alege o instanță de mai multe ori). Nu toate atributele instanțelor vor fi folosite în construirea arborilor, ci doar m dintre ele, alese aleator. Cel mai bun atribut dintre

cele m selectate va fi folosit pentru a continua dezvoltarea arborelui [BC]. Erorile din Random Forest sunt bazate pe două aspecte:

- Corelația dintre doi arbori diferiți;
- Puterea fiecărui arbore (puterea mai mare indicând o eroare mai mică).

Corelația și puterea pot fi ajustate în funcție de valoarea lui m. O valoare mică va scădea și cele două, în timp ce o valoare mare le va crește. Scopul este obținerea unei valori optime.

2.2.4 Extra Trees Classifier

Extra Trees este un algoritm similar cu Random Forest, de asemenea având drept idee construirea mai multor arbori de decizie. Spre deosebire de Random Forest, acest algoritm va utiliza întregul set de date de antrenament în formarea arborilor. De asemenea, atributele care determină modul de formare al arborilor sunt alese complet aleator [GEW06].

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

Figura 2.4: Algoritmul de separare pentru valori numerice [GEW06]

Modul de alegere al atributelor, dar și al separării instanțelor este prezentat în algoritmul din figura 2.4. Variabila n_{min} reprezintă numărul minim de instanțe necesare subsetului astfel încât să fie continuată separarea lor. Atunci când vine vorba

de problemele de clasificare, valoarea implicită a lui n_{min} este 2. Parametrul K determină numărul de părți în care se împarte subsetul în cadrul fiecărui nod. Valoarea implicită pentru K este $\sqrt{n}$, unde n reprezintă numărul de atribute.

Capitolul 3

Clasificarea datelor

3.1 Clasificarea ca metodă de învățare supervizată

Clasificarea este o metodă de învățare supervizată în care încercăm să încadrăm o instanță într-o clasă pe baza atributelor sale. Rezultatul furnizat de algoritm va fi eticheta care denumește clasa. Algoritmii de clasificare presupun astfel un mod de a separa diferitele etichete în funcție de atributele instanțelor. Instanțele au multiple atribute, acestea putând fi valori numerice, binare, imagini etc, dar au de asemenea o clasă. În funcție de numărul de clase existente în setul de date, putem distinge două tipuri de clasificare: clasificare binară, atunci când există doar două clase, și clasificare multi-clasă, atunci când există trei sau mai multe clase [Bur19].

Urmând pașii descriși în capitolul anterior, pentru a putea realiza un model de clasificare trebuie în primul rând să aplicăm transformări asupra datelor, pentru a le modifica într-un format care să permită obținerea unor rezultate mai bune. Diferitele metode de transformare ale datelor abordate în această lucrare sunt prezentate în subcapitolul următor.

Etapa următoare este reprezentată de evaluarea algoritmilor. Datele trebuie împărțite în date de antrenament și date de test. Datele de antrenament vor fi folosite pentru construirea modelului, în timp ce datele de test vor fi folosite pentru a estima performanța acestuia. Performanța unui model va fi stabilită pe baza unor metrici de performanță, care vor fi prezentate în următoarele subcapitole.

După obținerea unor rezultate preliminarii, putem opta pentru a îmbunătăți modelele care au obținut rezultate bune prin hiperparametrizarea acestora. După ce concluzionăm că am obținut rezultatul cel mai bun cu putință pe baza algoritmilor și a datelor asupra cărora am lucrat, vom putea salva modelul pentru a fi utilizat ulterior.

3.2 Transformarea datelor

Transformarea datelor presupune modificarea acestora într-un format care să permită construirea modelului. Anumiți algoritmi sunt scriși presupunând că setul de date urmează forma înțeleasă de aceștia și atunci rezultatul lor va fi afectat dacă datele nu prezintă trăsăturile corespunzătoare [Bro16].

3.2.1 Standardizarea datelor

Standardizarea datelor este o tehnică care transformă datele, astfel încât să urmeze o distribuție Gaussiană, să aibă media 0 și deviația standard 1. Acest mod de a transforma datele este în special de folos atunci când se utilizează algoritmi care necesită sau au o performanță sporită atunci când datele urmează distribuția Gaussiană, cum ar fi Regresia Liniară sau Linear Discriminate Analysis [Bro16].

Standardizarea mai poartă numele de normalizare „z-score", iar formula pentru z-score-ul/scorul standard al unei instanțe dintr-un feature j este următoarea:

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}}$$

unde x este valoarea (scorul) inițial, μ este media valorilor feature-ului înainte de standardizare, iar σ este deviația standard înainte de standardizare [Bur19].

3.2.2 Normalizarea datelor

Normalizarea datelor este un termen des folosit atunci când atributele setului de date sunt modificate astfel încât să fie încadrate în intervalul [0,1]. Această metodă este benefică pentru seturile de date sparse (care conțin multiple valori de 0) și este de asemenea utilă atunci când se utilizează algoritmul k-Nearest Neighbors [Bro16].

Definiția normalizării pentru clasa Normalizer din biblioteca scikit-learn presupune de asemenea transformarea fiecărei instanțe astfel încât să aibă norma 1. Spre deosebire de celelalte metode de transformare a datelor, conform documentației librăriei scikit-learn [slb], normalizarea se aplică asupra liniilor, nu coloanelor. Norma utilizată implicit este norma L2, cunoscută și ca norma euclidiană. În acest caz, normalizarea pentru linia j se va aplica astfel:

$$\hat{x}^{(j)} = \frac{x^{(j)}}{||x||_2}$$

unde x este valoarea inițială, iar $||x||_2$ este valoarea normei euclidiene, calculată cu formula următoare:

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

3.2.3 Scalarea datelor

Scalarea datelor este o metodă de tranformare a datelor care implică modificarea valorilor atributelor astfel încât acestea să fie în intervalul $[0, 1]$. Atributele sunt modificate independent una față de cealaltă. Aceasta este folositoare pentru algoritmi precum Gradient Descent și k-Nearest Neighbors [Bro16].

Scalarea datelor este reprezentată în biblioteca sci-kit learn de MinMaxScaler. Formula pentru scalarea min-max, cunoscută și ca normalizarea min-max, este:

$$\bar{x}^{(j)} = \frac{x^{(j)} - min^{(j)}}{max^{(j)} - min^{(j)}}$$

unde min și max sunt valoarea minimă, respectiv maximă a atributului j.

3.2.4 Binarizarea datelor

Binarizarea datelor se folosește de un prag în funcție de care sunt modificate valorile: dacă valoarea depășește pragul va fi înlocuită cu 1, altfel va fi înlocuită cu 0. Binarizarea datelor este folositoare atunci când se dorește adăugarea ulterioară a altor atribute [Bro16].

3.3 Metrici de performanță

Pentru a evalua eficiența unui model, vom utiliza metrici de performanță aplicate asupra datelor de test. Atunci când vine vorba de clasificare, există mai multe opțiuni pentru metrici precum: acuratețea în clasificare, pierderea logaritmică, matricea de confuzie și aria de sub curba ROC și altele. Alegerea metricii depinde de tipul de problemă de clasificare (binară, multiclass), dar și de setul de date.

3.3.1 Matricea de confuzie

Matricea de confuzie este o reprezentare a predicțiilor realizate de model. Liniile reprezintă predicțiile, în timp ce coloanele reprezintă adevăratele clase [Bro16]. Astfel, pe linia x și coloana y vom avea numărul de instanțe considerate de model ca aparținând clasei x, dar care de fapt aparțin clasei y. În cazul problemelor de clasificare binară, vom avea:

$$\begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

unde TP este numărul valorilor adevărat pozitive („true positive”), FN este numărul valorilor fals negative („false negative”), FP este numărul valorilor fals pozitive

(„false positive"), iar TN este numărul valorilor adevărat negative („true negative") [Bur19]. Valorile adevărat pozitive, respectiv negative, reprezintă cele corect încadrate de către model ca aparținând claselor 0, respectiv 1, în timp ce cele false reprezintă încadrarea incorectă la clasele 0, respectiv 1.

Matricea de confuzie este benefică și atunci când problema are multiple clase, deoarece ne permite să vedem câte date sunt incorect clasificate ca fiind parte dintr-o altă clasă. În acest caz pot fi adăugate mai multe date pentru a face distingerea dintre cele două clase mai ușor de observat de către model, sau să adăugăm atribute care să facă diferența dintre clase [Bur19].

3.3.2 Acuratețea

Acuratețea de clasificare reprezintă raportul dintre numărul de preziceri corecte și numărul de preziceri totale. Această metrică este benefică atunci când există un număr egal de instanțe în fiecare clasă [Bro16]. Atunci când ne folosim de acuratețe presupunem că toate erorile de clasificare sunt la fel de grave, lucru care nu va fi întotdeauna adevărat, motiv pentru care utilizarea acestei metrici trebuie făcută cu mare atenție asupra atributelor datelor și asupra problemei în sine [Bur19]. Acuratețea se calculează folosind formula următoare:

$$\text{acuratețe} = \frac{TP + TN}{TP + FP + TN + FN}$$

3.3.3 Pierderea logartimică

Pierderea logaritmică, cunoscută și ca pierdere logistică sau pierdere „cross-entropy" este o metrică de clasificare care evaluează probabilitățile estimate de către model în ceea ce privește apartenența unei instanțe la o clasă. Aceasta este calculată în funcție de cât de apropiate sau depărtate au fost estimările modelului față de adevăr. O pierdere logaritmică de 0 presupune un model care a evaluat corect toate datele, iar cu cât se depărtează valoarea de 0, scade corectitudinea modelului [Bro16]. Pierderea logaritmică se calculează folosind formula următoare:

$$l = -\frac{1}{N}\sum_{i=1}^{N}(y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i))$$

unde N este numărul de instanțe, y_i este rezultatul clasa binară pentru instanța i iar p_i este probabilitatea obținută de algoritm ca instanța i să aparțină clasei 1 [DP].

3.3.4 Aria de sub curba ROC

Aria de sub curba ROC poate fi utilizată atunci când vine vorba de probleme de clasificare binară. Ea reprezintă capacitatea modelului de a distinge între cele două

clase. Valoarea 1 implică un model capabil de a realiza toate prezicerile corecte, iar cu cât scade devine mai îndoilenic [Bro16]. Curba ROC este determinată de rata valorilor adevărat pozitive și de rata valorilor fals pozitive. Cele două valori se calculează astfel:

$$\text{rata valorilor adevărat pozitive} = \frac{TP}{TP + FN};$$
$$\text{rata valorilor fals pozitive} = \frac{FP}{FP + TN}.$$

Capitolul 4

Aplicaţia practică

În cadrul acestei lucrări au fost realizate atât o aplicație Web, cât și un model de clasificare care să fie utilizat în cadrul aplicației. Aplicația web este alcătuită din două părți: componenta client scrisă în TypeScript, folosind framework-ul Ionic și librăriile React și axios, și componenta server, folosind limbajul Python, împreună cu librăriile sci-kit learn și pickle și framework-ul Flask. Modelul a fost construit de asemenea în limbajul Python, folosind aceleași librării.

4.1 Prezentare generală

Aplicația *HeartVitality* este o aplicație Web, compusă dintr-o componentă client, scrisă în TypeScript, și o componentă server, scrisă în Python. Scopul acesteia este de a permite unui utilizator să obțină pe baza unor date care pot fi obținute și în afara unui control medical un rezultat în ceea ce privește posibilitatea existenței unei boli cardiovasculare. Astfel, accesibilitatea a fost o componentă cheie în timpul realizării aplicației. *HeartVitality* dispune de mai multe funcționalități, acestea fiind următoarele:

- completarea formularului pentru diagnosticul cardiovascular: la deschiderea aplicației, utilizatorul are posibilitatea de a completa un formular care presupune trecea unor date semnificative pentru obținerea unui rezultat;
- obținerea unui diagnostic: prin utilizarea unui model de clasificare, implicit cel obținut în secțiunea anterioară utilizând algoritmul Gradient Boosting, un utilizator obține pe baza datelor furnizate de acesta un rezultat pozitiv sau negativ în ceea ce privește riscul existenței unei boli cardiovasculare. Acest lucru are loc doar dacă datele introduse de utilizator sunt valide/complete;
- obținerea unui raport în format PDF: după ce utilizatorul primește rezultatul, prin completarea câmpurilor pentru prenume și numele de familie îi este

oferită posibilitatea de a descărca un document PDF care va conține numele, datele introduse în formular, rezultatul obținut și data la care s-a realizat formularul;

- modificarea limbii aplicației: pagina principală (cea care conține formularul) dispune de butoane pentru limba română și engleză care odată apăsate permit modificarea limbii în funcție de nevoile utilizatorului;
- autentificarea ca administrator: pentru a accesa funcționalitățile exclusive administratorilor, aceștia sunt nevoiți ca măsură de securitate să introducă o pereche validă de nume de utilizator și parolă;
- construirea și testarea modelelor de clasificare: administratorii au acces la o pagină specială care le pune la dispoziție mai mulți algoritmi, metode de transformare a datelor și în situații speciale hiperparametrizarea. Astfel, ei pot să construiască modele noi de clasificare și să vizualizeze mai multe metrici obținute în urma testării lor;
- modificarea modelului utilizat: modelul implicit poate fi înlocuit, după nevoie, de un nou model realizat de către un administrator. Modelul implicit va rămâne salvat astfel încât să se poată anula modificarea fără a îl reconstrui.

În ceea ce privește rularea aplicației, aceasta este menită pentru utilizarea de pe un calculator personal. Pentru server utilizatorul are nevoie de versiunea de Python 3.9 și de librăriile scikit-learn, pickle, iar pentru client sunt necesare librăriile React, axios și jsPDF, dar și de framework-ul Ionic. Este necesar de asemenea un browser web precum Opera sau Google Chrome, alegerea fiind la latitudinea utilizatorului.

4.2 Arhitectura aplicației

HeartVitality este un proiect client/server, așadar vom avea două aplicații care vor comunica între ele: aplicația client și aplicația server. Comunicarea între cele două se bazează pe cereri HTTP. Clientul trimite cereri către server, iar acesta va procesa cererea și va trimite răspunsul adecvat.

Spre exemplu, atunci când utilizatorul apasă butonul de trimitere a datelor din formular, se realizează o cerere de tip GET către '/api/result', care va avea ca parametri datele din formular. Înainte de a trimite un răspuns va fi efectuată o verificare asupra datelor pentru a verifica validitatea lor. Dacă au ajuns date incorecte către server va fi trimis înapoi un răspuns cu cod 400 (”BAD REQUEST”) care va conține un mesaj de eroare. Dacă datele trimise sunt corecte, modelul de clasificare le va evalua, iar predicția sa va fi trimisă ca un răspuns cu cod 200 (”OK”).

Pentru a realiza aplicația într-un mod eficient și pentru a separa părțile care țin de client de cele care țin de server, aplicația în întregime are o structură de fișiere strictă, unde elementele sunt grupate în funcție de rolul în cadrul aplicației. Fișierele generate automat care nu au fost modificate nu vor fi menționate.

4.2.1 Client

Componenta client a aplicației *HeartVitality* a fost realizată utilizând limbajul de programare TypeScript. Acest limbaj reprezintă o versiune a limbajului JavaScript care permite de asemenea acordarea de tipuri de date obiectelor. Avantajul este dat de faptul că programatorul poate să ofere datelor un tip explicit care să asigure depistarea utilizării incorecte a variabilelor/constantelor [Typ]. Împreună cu TypeScript, am utilizat de asemenea următoarele framework-uri și librării:

- framework-ul Ionic;
- librăria React;
- librăria axios;
- librăria jsPDF.

Framework-ul Ionic este utilizat pentru realizarea UI-ului. El este cunoscut pentru portabilitatea sa, același cod putând fi utilizat pentru a realiza o aplicație Web, Android sau iOS. Un alt avantaj al lui Ionic este compatibilitatea sa cu framework-uri și librării precum Angular, React și Vue [Ion]. Toate elementele de UI din aplicație au fost construite folosind componente Ionic.

Librăria React este utilizată pentru a construi componente, de la părți dintr-o pagină până la pagini propriu-zise [Rea]. Această librărie ne oferă acces și la hook-uri, care permit vizualizarea modificării datelor în cadrul paginii. React de asemenea ne permite ca dezvoltatori să vizualizăm modificările efectuate asupra paginilor fără să fie necesară oprirea execuției programului. Toate paginile din aplicație sunt componente React care conțin în interiorul lor hook-uri precum useEffect() și useState().

Axios este o librărie utilizată pentru a eficientiza trimiterea de cereri HTTP din partea client-ului, iar jsPDF este o librărie cu ajutorul căreia putem crea și descărca documente în format PDF utilizând strict cod JavaScript.

Aplicația client este conținută în folderul intitulat „frontend”. La rândul său, el dispune de următoarele foldere:

1. public: conține un folder images unde sunt localizate imaginile în format JPG pentru butoanele pentru limbă

2. src: acest folder va conține codul propriu-zis, și va fi format din următoarele foldere și fișiere:

    - classes: un folder care va conține clasele necesare client-ului (spre exemplu clasa FormInfo care reține datele introduse în formular și rezultatul obținut de model);
    - pages: folder-ul acesta va conține fișierele TypeScript aferente paginilor și opțional fișiere CSS pentru a modifca aspectul unor componente din cadrul paginilor. Fișierele sunt separate în foldere pentru fișiere TypeScript, respectiv CSS;
    - App.tsx: acest fișier TypeScript are rolul de a furniza aplicației rutele care accesează paginile și pagina principală (default).

4.2.2 Server

Componenta server a aplicației *HeartVitality* a fost formată cu ajutorul limbajului de programare Python, același limbaj utilizat și pentru dezvoltarea modelului. Motivul pentru care am optat pentru acest limbaj de programare pentru dezvoltarea server-ului și nu alte limbaje precum Java sau C# este familiaritatea obținută pe parcursul dezvoltării modelului în ceea ce privește apelarea funcțiilor și claselor furnizate de scikit-learn. Formatul în care a fost salvat modelul cu ajutorul librăriei pickle ar necesita conversii pentru a putea fi utilizat în alte limbaje.

Librăria scikit-learn este specializată pe învățarea automată, oferind acces la clase pentru fiecare algoritm de clasificare și de ansamblu prezentat în această lucrare, funcții pentru testarea algoritmilor etc. Pickle a fost folosit pentru stocarea modelelor create de administrator sub formă de fișiere .pkl, dar și pentru accesarea modelelor stocate în ele și în fișierul pentru modelul implicit. Astfel, nu este necesar să reconstruim modelele de fiecare dată când pornim server-ul. Am utilizat de asemenea framework-ul Flask. Flask este un framework care permite dezvoltarea de aplicații web cu ajutorul limbajului Python. Cu ajutorul său putem realiza un server unde să atribuim funcții diferitelor URL-uri.

Aplicația server se află în folderul intitulat ,,backend". La rândul său, ea dispune de următoarele foldere:

1. main: conține fișierul app.py care va inițializa server-ul și unde vor fi așteptate și procesate cererile HTTP pe parcursul rulării aplicației

2. resoruces: acest folder va conține fișiere care nu reprezintă cod, dar care conțin informații relevante pentru server:

- authentication: acest folder va conține un fișier text cu toate perechile nume de utilizator/parolă care vor fi acceptate de pagina de login și un fișier text cu token-ul asociat fiecărui cont.
- languages: un folder care va conține pentru fiecare limbă acceptată de aplicație câte un folder. Aceste foldere sunt formate din fișiere text pentru limba respectivă, fiecare fișier text conținând textul paginii căreia îi poartă numele.
- models: folder-ul acesta va conține modelul inițial (model.pkl) care va fi utilizat implicit în cadrul aplicației, iar în cazul creării de noi modele în pagina de administrator, ultimul creat va fi de asemenea salvat aici;

3. test: un folder care va conține teste efectuate asupra funcțiilor pentru a asigura funcționarea corespunzătoare a acestora;
4. validation: funcțiile din folder-ul acesta vor efectua validări asupra datelor primite de la client și vor dicta răspunsul oferit.

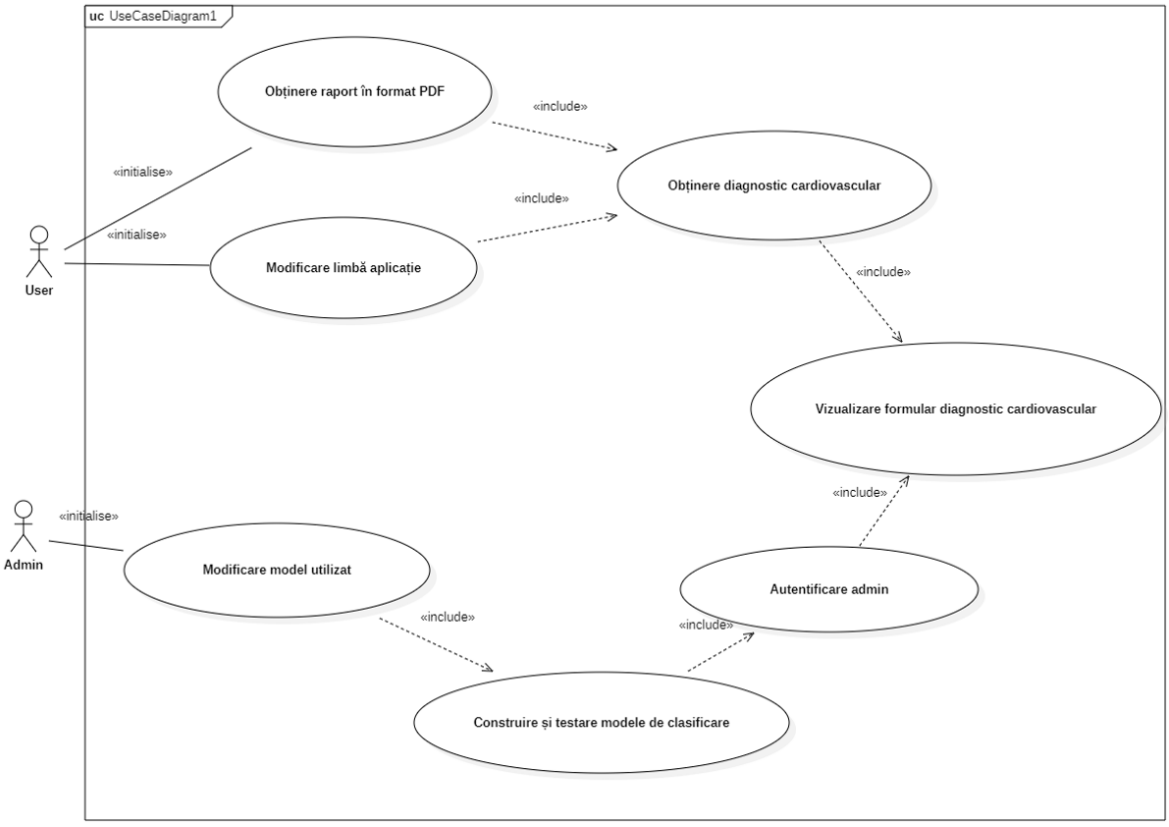

Figura 4.1: Diagrama UML a cazurilor de utilizare

Diagrama din figura 4.1 poate fi asociată funcționalităților menționate la început, dar de asemenea ne arată și modul în care paginile aplicației vor comunica între ele. Aplicația dispune de următoarele patru pagini:

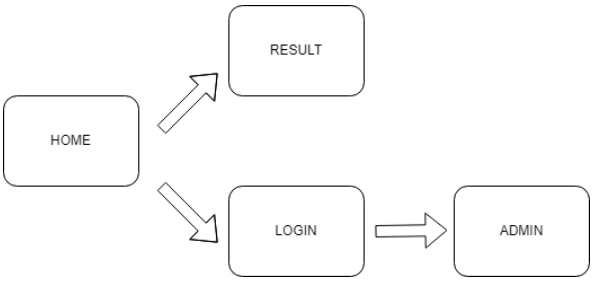

Figura 4.2: Modul de parcurgere al paginilor

- Home ('/home'): această pagină va conține formularul, opțiunea de schimbare a limbii și butoanele pentru trimiterea datelor și autentificarea ca administrator;
- Result ('/result'): această pagină va conține rezultatul obținut de modelul de clasificare, câmpurile pentru completarea numelui și butonul care va construi și descărca raportul PDF;
- Login ('/login'): această pagină va conține câmpurile pentru numele de utilizator și parolă care trebuie completate de administrator pentru a se autentifica;
- Admin ('/admin'): această pagină va permite administratorului să construiască noi modele de clasificare, să vizualizeze mai multe metrici ale acestora și să înlocuiască modelul implicit cu cel nou obținut.

Având aceste patru pagini, modul în care acestea vor fi accesate este prezentat în figura 4.2. Dacă utilizatorul aplicației încearcă să acceseze altă pagină care necesită informații din pagina curentă prin modificarea link-ului, el va fi redirecționat la pagina corectă. Spre exemplu, dacă utilizatorul va completa și încerca să intre din start pe pagina Result, el va fi trimis la pagina Home.

4.2.3 Manual de utilizare

Atunci când utilizatorul accesează aplicația, el va fi trimis la pagina principală a acesteia, reprezentată de figura 4.3. Această pagină conține formularul, alături de trei butoane: două pentru limba engleză, respectiv română în colțul stânga-sus, și un altul pentru autentificarea ca administrator în colțul de dreapta sus. Apăsarea butonului pentru limbă va schimba textul în limba respectivă dacă nu era deja varianta aleasă.

Pentru un simplu utilizator, scopul aplicației este e de a prelua datele importante pentru a stabili riscul existenței unei boli cardiovasculare. Pentru a furniza datele utilizatorul trebuie, în funcție de caz, să selecteze o opțiune sau să completeze un câmp pentru fiecare parte din formular. Câmpurile (înălțimea, greutatea și cele două

tipuri de tensiuni) vor accepta doar valori numerice. Sub formular se află un buton pentru trimiterea datelor. Dacă există părți din formular unde au fost introduse valori imposibile (precum o zi de naștere din viitor, înălțime negativă etc.) sau nu au fost introduse date sau selectată o opțiune la apăsarea butonului de trimitere a datelor, va fi afișat un mesaj reprezentativ pentru părțile incorecte din mesaj. Spre exemplu, dacă câmpul pentru greutate conține o valoare negativă și pentru tensiunea sistolică nu a fost completată o valoare, mesajul afișat va fi „Greutatea nu poate fi negativă. Câmpul pentru tensiunea sistolică nu a fost completat.”.

Figura 4.3: Pagina principală a aplicației cu limba română (varianta implicită)

Dacă datele introduse în formular sunt valide și complete, utilizatorul va fi redirecționat către o nouă pagină unde va fi afișat un text în funcție de rezultatul obținut de modelul de clasificare, negativ sau pozitiv. Sub acest text, utilizatorul va avea opțiunea de a completa două câmpuri, unul pentru prenume și unul pentru nume. Dacă completarea acestora, utilizatorul poate apăsa pe butonul de dedesubt care va realiza și descărca un document în format PDF care va conține numele utilizatorului, datele furnizate de acesta, rezultatul determinat de model și data la care a fost creat documentul.

Administratorii aplicației au opțiunea de a se autentifica pentru a accesa funcționalități exclusive. Apăsarea butonului de login din colțul din dreapta-sus al paginii principale va trimite administratorul la o nouă pagină unde va avea de completat numele de utilizator și parola aferentă. După ce a fost introdus un cont corect și a fost apăsat butonul de confirmare, administratorul va fi redirecționat către o nouă pagină.

În cadrul paginii la care a fost realizată redirecționarea, utilizatorul va putea să își creeze un nou model de clasificare, fiindu-i oferite opțiuni în ceea ce privește algoritmul care să fie utilizat (cei prezentați în capitolul *Învățare automată*), modul

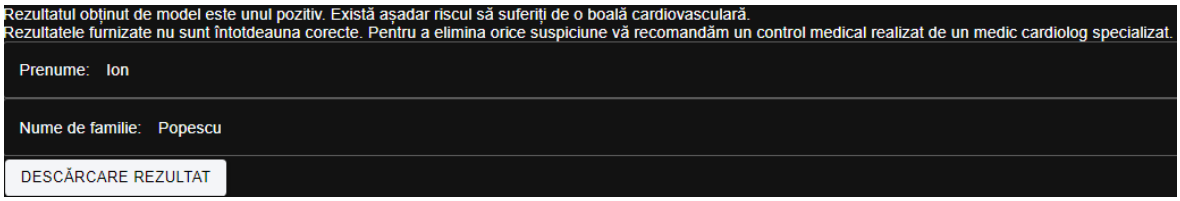

Figura 4.4: Rezultatul furnizat de model, împreună cu opțiunea de a completa câmpurile cu nume pentru realizarea PDF-ului

de preprocesare al datelor utilizate pentru antrenare (dintre modalitățile prezentate în secțiunea *Transformarea datelor* din cadrul capitolului 3), iar dacă algoritmul este unul dintre cei pentru care s-a testat hiperparametrizarea parametrilor în subcapitolul următor, se poate opta pentru realizarea acesteia. După ce administratorul își configurează caracteristicile dorite pentru noul model și apasă butonul de confirmare, vor fi afișate metrici ale modelului rezultat pe baza celor prezentate în partea de *Metrici de clasificare*. Dacă rezultatele obținute sunt mulțumitoare, utilizatorul poate alege să fie înlocuit modelul curent cu cel nou. Se poate alege de asemenea revenirea la modelul inițial în cazul în care noul model nu are performanța dorită.

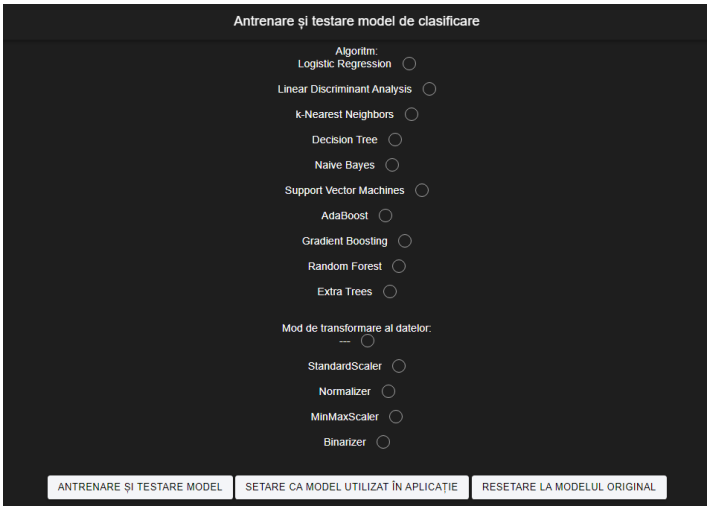

Figura 4.5: Opțiunile pentru generarea modelului oferite administratorului

4.3 Rezultate experimentale

4.3.1 Condiții de testare

Un model de învățare automată este rezultatul antrenării unui algoritm de învățare automată utilizând date. Dat fiind faptul că problema este una de clasificare, modelul va fi capabil, cu o anumită acuratețe, să clasifice datele primite într-o categorie, folosind informațiile acumulate în urma pregătirii acestuia cu datele inițiale. [GS13]

Modelul rezultat în urma acestui studiu este rezultatul testării mai multor algoritmi de clasificare și a unor metode de transformare a datelor inițiale, verificând care dintre acestea au potențial și cum pot fi îmbunătățite mai departe.

Pentru a realiza modelul de clasificare am folosit limbajul de programare Python și librăriile sci-kit learn și pickle. Python este des asociat cu inteligența artificială, unul dintre motive fiind multitudinea de librării disponibile asociate cu acest domeniu.

Setul de date iniţial are 70 de mii de înregistrări şi 11 atribute (toate valori numerice: întregi, reale sau binare) plus atributul ,,target", o valoare binară care indică dacă persoana în cauză suferă sau nu de o boală cardiovasculară. În mod concret, atributele fiecărei instanțe sunt prezentate în tabelul 4.1.

Atribut	Reprezentare
Vârstă (age)	număr întreg
Genul (gender)	1- feminin, 2 - masculin
Înălțime (height)	număr întreg
Greutate (weight)	număr întreg
Tensiunea sistolică/mare (ap_hi)	număr întreg
Tensiunea diastolică/mică (ap_lo)	număr întreg
Cantitatea de colesterol (cholesterol)	1 – normală, 2 – uşor ridicată, 3 – ridicată
Cantitatea de glucoză (gluc)	1 – normală, 2 – uşor ridicată, 3 – ridicată
Fumatul activ (smoke)	0 – nu fumează activ, 1 – fumează activ
Consumul de alcool (alco)	0 - consum normal, 1 - consum ridicat
Activitate fizică (active)	0 - lipsă/scăzută, 1 - regulată
Boală cardiovasculară (cardio)	0 - negativ, 1 - pozitiv

Tabela 4.1: Feature-urile setului de date și reprezentarea acestora

Pentru a putea vedea modul în care sunt împărțite diferitele valori pentru fiecare atribut, am realizat folosind funcția ,,plot" din librăria Numpy o histogramă. Aceasta, reprezentată în figura 4.6, prezintă numărul şi dimensiunea diferitelor valori care pot fi luate de către fiecare dintre atribute. Deoarece majoritatea atributelor pot avea un număr limitat (2-3) de valori, nu se poate observa o diversitate mare decât în ceea ce priveşte atributele referitoare la vârstă, înălţime şi greutate. Deşi tensiunea sistolică (mare) şi cea diastolică (mică) pot avea de asemenea mai multe valori diferite, acestea sunt în general suficient de apropiate pentru a nu se poate observa o diferenţă majoră. Faptul că numărul persoanelor care prezintă boli cardiace (35021) este foarte apropiat de numărul indivizilor sănătoși (34979) indică faptul că acuratețea este o metrică de clasificare pe care o putem utiliza cu încredere.

4.3.2 Criterii de evaluare

Pentru a avea un rezultat mai eficient decât ce am obține prin simpla împărțire

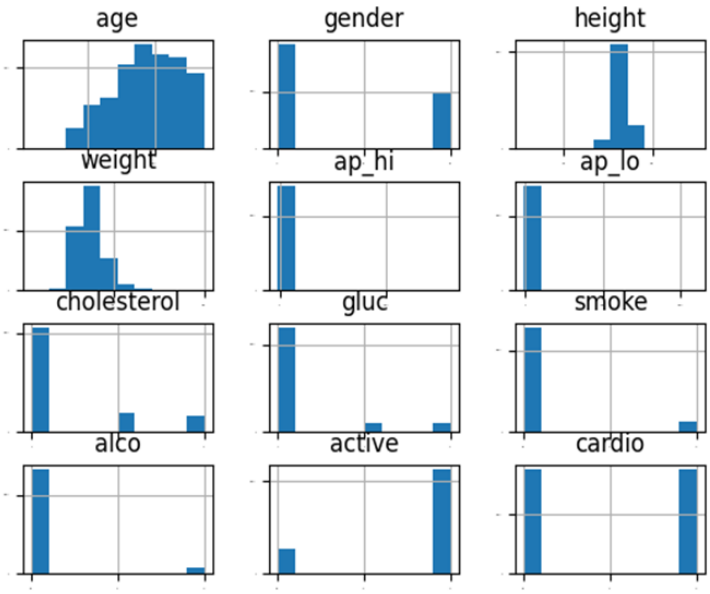

Figura 4.6: Reprezentarea sub formă de histogramă a setului de date

în date de antrenament și date de test, va fi utilizată de asemenea validarea k-fold cross. Acest mod de validare presupune utilizarea unei valori k pentru a împărți setul de date în k părți denumite fold-uri. Dintre acestea, $k-1$ sunt utilizate pentru antrenament și cea rămasă pentru testare. Aceast proces este repetat astfel încât fiecare fold să poată fi utilizat în testare. Pentru această problemă am ales pentru k valoarea 10, una recomandată pentru seturile de date de această dimensiune [Bro16].

Media celor 10 teste ne va estima rezultatul pentru un număr restrâns de cazuri. Deoarece dorim să avem un rezultat care să acopere o populație mai amplă, vom utiliza intervale de încredere. Deși nu vom avea o valoare exactă, vom putea încadra rezultatul corect într-un interval cu o corectitudine stabilită de noi. Dat fiind că avem un număr mic de valori (10) asupra cărora vom dori să calculăm intervalul de încredere ne vom folosi de formula următoare:

$$interval = \bar{x} \pm t\frac{s}{\sqrt{n}}$$

unde $\bar{x}$ este media observațiilor, n este numărul de observații, t este valoarea critică bazată pe nivelul de încredere dorit (în acest caz va fi n-1) și s este deviația standard a observațiilor [dat].

Am testat astfel cei şase algoritmi de modelare (Regresia Logistică, Linear Discriminant Analysis, K-Nearest Neighbors, Arborii de Decizie, Naïve Bayes şi Support Vector Machine) folosind clasele aferente oferite de biblioteca scikit-learn. Pentru

a determina acuratețea algoritmilor, am făcut media aritmetică a celor 10 rezultate obținute în urma validării k-fold cross.

Conform tabelei 4.2, se poate observa faptul că algoritmul de Regresie Logistică are cea mai mare acurateţe, apropiată de 70%. Un alt algoritm, K-Nearest Neighbors, a atins o acurateţe apropiată. Deoarece ceilalți algoritmi pot suferi din cauza formatului actual al datelor, este necesar să efectuăm încă un test, cu datele transformate.

Algoritm	Acuratețe
Regresie Logistică	69,80% $\pm$ 0.009%
Linear Discriminant Analysis	64,68% $\pm$ 0.008%
k-Nearest Neighbors	68,20% $\pm$ 0.012%
Arbori de decizie	63,27% $\pm$ 0.013%
Naive Bayes	58,98% $\pm$ 0.014%
Support Vector Machines	60,49% $\pm$ 0.013%

Tabela 4.2: Rezultatele algoritmilor de clasificare cu datele în forma inițială (în urma realizării a 10 teste folosind k-fold)

Am utilizât mai mulţi algoritmi de transformare (standardizarea, normalizarea, scalarea și binzarizarea), din nou utilizând clasele aferente din librăria scikit-learn pentru a vedea dacă acurateţea poate fi îmbunătăţită. Aceste transformări pot să crească sau să scadă performanța algoritmilor de clasificare, în funcție de nevoile acestora referitoare la date. Pentru a asigura faptul că transformările se aplică doar asupra fold-urilor de antrenament am utilizat clasa Pipeline din biblioteca sci-kit learn. Aceasta asigură că datele de test vor fi intacte și că nu vor influența transformarea datelor de antrenament pentru a garanta obținerea unui rezultat corect [Bro16]. Așadar, am creat un pipeline pentru fiecare pereche algoritm de transformare - algoritm de clasificare și am comparat din nou media valorilor de acuratețe a celor 10 împărțiri în fold-uri.

Algoritm	Acuratețe	
	Standardizare	Normalizare
Regresie Logistică	72,12% $\pm$ 0.007%	63,23% $\pm$ 0.039%
Linear Discriminant Analysis	64,68% $\pm$ 0.008%	66,11% $\pm$ 0.010%
k-Nearest Neighbors	65,19% $\pm$ 0.015%	68,03% $\pm$ 0.007%
Arbori de Decizie	63,26% $\pm$ 0.011%	63,43% $\pm$ 0.004%
Naive Bayes	59,02% $\pm$ 0.014%	56,63% $\pm$ 0.013%
Support Vector Machine	72,87% $\pm$ 0.006%	50,98% $\pm$ 0.006%

Tabela 4.3: Rezultatele algoritmilor de clasificare pentru standardizare și normalizare (în urma realizării a 10 teste folosind k-fold)

Conform tabelelor 4.3 și 4.4, se poate observa ca standardizarea a avut un impact pozitiv asupra majorității algoritmilor, k-Nearest Neighbors fiind singurul căruia i-

Algoritm	Acuratețe	
	Scalare	Binarizare
Regresie Logistică	64,79% ± 0.008%	51,43% ± 0.011%
Linear Discriminant Analysis	64,68% ± 0.008%	51,43% ± 0.011%
k-Nearest Neighbors	60,34% ± 0.013%	49,18% ± 0.014%
Arbori de Decizie	63,15% ± 0.013%	51,53% ± 0.011%
Naive Bayes	59,02% ± 0.014%	50,35% ± 0.007%
Support Vector Machine	64,19% ± 0.008%	51,53% ± 0.011%

Tabela 4.4: Rezultatele algoritmilor de clasificare pentru scalare și binarizare (în urma realizării a 10 teste folosind k-fold)

a scăzut acuratețea. Support Vector Machines, penultimul algoritm ca performanță înainte de transformarea datelor, are acum cea mai mare acuratețe, apropiată de 73%, o valoare cu peste 12% mai mare decât înainte. Regresia Logistică are de asemenea o acuratețe de peste 72%, așadar vom lua în considerare cei doi algoritmi ca posibili candidați pentru cel utilizat în modelul final. În ceea ce privește celelalte metode de transformare putem spune că nu am avut rezultate cu un impact pozitiv asemănător, iar binarizarea în special a avut un impact negativ asupra tuturor rezultatelor.

În continuare, vom efectua aceleași două verificări pentru algoritmii de tip ansamblu (AdaBoost, Gradient Boosting, Random Forest, Extra Trees) pentru a vedea cum se comportă pentru aceleași date.

Algoritm	Acuratețe
AdaBoost	73,04% ± 0.006%
Gradient Boosting	73,62% ± 0.006%
Random Forest	71,48% ± 0.008%
Extra Trees	70,28% ± 0.010%

Tabela 4.5: Rezultatele algoritmilor de tip ansamblu cu datele în forma inițială (în urma realizării a 10 teste folosind k-fold)

Algoritm	Acuratețe	
	Standardizare	Normalizare
AdaBoost	73,04% ± 0.006%	72,00% ± 0.006%
Gradient Boosting	73,63% ± 0.006%	73,26% ± 0.005%
Random Forest	71,55% ± 0.009%	71,65% ± 0.007%
Extra Trees	70,28% ± 0.010%	70,21% ± 0.008%

Tabela 4.6: Rezultatele algoritmilor de tip ansamblu pentru standardizare și normalizare (în urma realizării a 10 teste folosind k-fold)

Așa cum reiese din tabelele 4.5, 4.6 și 4.7, algoritmul de tip ansamblu cel mai eficient este Gradient Boosting, având o acuratețe de aproximativ 74% în majoritatea cazurilor. Scalarea și standardizarea au fost metodele de transformare cele mai

Algoritm	Acuratețe	
	Scalare	Binarizare
AdaBoost	73,04% $\pm$ 0.006%	51,43% $\pm$ 0.011%
Gradient Boosting	73,63% $\pm$ 0.006%	51,53% $\pm$ 0.011%
Random Forest	71,43% $\pm$ 0.008%	51,53% $\pm$ 0.011%
Extra Trees	70,23% $\pm$ 0.010%	51,53% $\pm$ 0.011%

Tabela 4.7: Rezultatele algoritmilor de tip ansamblu pentru scalare și binarizare (în urma realizării a 10 teste folosind k-fold)

utile pentru acest algoritm. Gradient Boosting devine astfel cel de-al treilea candidat pentru algoritmul care va fi utilizat spre realizarea modelului. Vom lua în considerare de asemenea un al patrulea algoritm, AdaBoost, cu acuratețea de 73,04% pentru aceleași tipuri de transformări. În general, transformarea datelor nu a avut un impact pozitiv asupra eficienței, acuratețea rămânând adesea aceeași sau mai mică. Există totuși cazuri în care aceasta s-a îmbunătățit, dar nu semnificativ. Binarizarea, din nou, a afectat într-o manieră negativă rezultatele algoritmilor.

Pasul final în ceea ce privește alegerea algoritmului pentru realizarea modelului îl reprezintă ajustarea parametrilor algoritmilor luați în considerare. Acest proces mai poartă numele de optimizare a hiperparametrilor [Bro16]. Algoritmii luați în considerare în acest pas sunt Regresia Logistică, Support Vector Machines, Gradient Boosting și AdaBoost. Biblioteca scikit-learn ne oferă două metode de ajustare a parametrilor: căutare în rețea, care se bazează pe utilizarea fiecărei combinații posibile de parametri aflați într-o rețea, și căutarea aleatorie, care își va alege pentru un număr fix de iterații combinații de parametri. Am optat pentru căutarea în rețea deoarece acoperă fiecare combinație posibilă și nu vom putea rata cea mai bună alegere posibilă.

În ceea ce privește Regresia Logistică, parametrii luați în considerare au fost $penalty$ și $solver$. Parametrul $penalty$ reprezintă norma penalizării, și are ca valoare implicită $l2$. Celelalte valori posibile sunt $None$, $l1$ și $elasticnet$, care presupune aplicarea ambelor penalizări. $solver$ determină optimizarea algoritmului, având mai multe valori posibile. Singura valoare pentru $solver$ compatibilă cu toate valorile posibile pentru $penalty$ este $saga$. Dintre celelalte valori, $liblinear$ este compatibilă doar cu $l1$ și $l2$, iar restul cu $l2$ și $None$. Metoda de transformare aleasă este standardizarea.

Pentru algoritmul Support Vector Machines am optat pentru modificarea parametrilor C și $kernel$. Conform documentației clasei SVC oferite de scikit-learn, C reprezintă parametrul de regularizare, este o valoare reală strict pozitivă, iar puterea regularizării este invers proporțională cu aceasta. $kernel$ determină tipul de kernel utilizat. Tipul de transformare al datelor este standardizarea.

Gradient Boosting, algoritmul cu cea mai bună acuratețe până în acest punct, are

o multitudine de parametri. În scopul optimizării algoritmului am optat pentru trei dintre aceștia: $n\_estimators$, $learning\_rate$ și max_depth. $n\_estimators$ va reprezenta numărul de pași efectuați de algoritm. $learning\_rate$ este un număr real care desemnează cu cât este scăzută contribuția fiecărui arbore. max_depth poate lua valori întregi sau valoarea $None$ și va reprezenta numărul de noduri ale arborelui, iar dacă este $None$, nodurile vor fi dezvoltate până când toate frunzele sunt pure sau conțin mai puține instanțe decât $min\_samples\_split$. Spre deosebire de cei doi algoritmi precedenți, la Gradient Boosting am obținut același rezultat pentru standardizare și scalare. Hiperparametrizarea va fi testată astfel folosind ambele metode de transformare.

În ceea ce privește AdaBoost, acesta are anumiți parametri în comun cu Gradient Boosting. Având în minte acest considerent, vom folosi din nou $n\_estimators$ și $learning\_rate$. În ceea ce privește metoda de transformare a datelor, am urmat același principiu ca în cadrul Gradient Boosting.

4.3.3 Rezultatele finale

Algoritm	Parametri optimi	Acuratețe
Regresie Logistică	$penalty = none$, $solver = newton - cg$	72,14% $\pm$ 0.006%
Support Vector Machines	$C = 2.0$, $kernel = rbf$	72,97% $\pm$ 0.006%
Gradient Boosting + standardizare	$n\_estimators = 100$, $learning\_rate = 0.1$, $max\_depth = 3$	73,63% $\pm$ 0.006%
Gradient Boosting + scalare	$n\_estimators = 100$, $learning\_rate = 0.1$, $max\_depth = 3$	73,63% $\pm$ 0.006%
AdaBoost + standardizare	$n\_estimators = 500$, $learning\_rate = 0.5$	73,09% $\pm$ 0.006%
AdaBoost + scalare	$n\_estimators = 500$, $learning\_rate = 0.5$	73,09% $\pm$ 0.006%

Tabela 4.8: Rezultatele algoritmilor în urma optimizării hiperparametrilor (în urma realizării a 10 teste folosind k-fold)

Rezultatele procesului de optimizare pot fi observate în tabela 4.8. După cum se poate observa, comparând valorile cu cele din tabelele precedente, optimizarea nu a îmbunătățit semnificativ acuratețea. Gradient Boosting nu a avut nevoie de optimizare, dat fiind că valoarile optime ale parametrilor au fost cele implicite. Deoarece are în continuare cel mai bun rezultat, Gradient Boosting va fi algoritmul utilizat pentru realizarea modelului. Având aceeași acuratețe atât pentru standardizare, cât

și pentru scalare, putem folosi oricare dintre cele două metode de transformare a datelor. Modelul din aplicație utilizează scalarea. Folosind librăria pickle putem salva modelul într-un fișier pentru a-l utiliza în aplicație fără a fi nevoie să fie generat de fiecare dată.

Capitolul 5

Concluzii

5.1 Rezultate

În urma testării mai multor algoritmi de clasificare, cât și a mai multor metode de transformare a datelor, am obținut o acuratețe de aproximativ 73,63% utilizând algoritmul Gradient Boosting. Acest rezultat este unul satisfăcător atunci când luăm în considerare că datele pot fi obținute de acasă, fără a apela la un control medical, presupunând că individul dispune de aparatura necesară pentru măsurarea tensiunii, colesterolului și a glicemiei.

Aplicația web a fost creată din dorința de a fi ușor accesibilă utilizatorilor. Menținerea unui format accesibil și opțiunea pentru selectarea limbii au contribuit la acest lucru. Realizarea celor două componente ale aplicației client-server nu a fost una dificilă, iar realizarea conexiunii între cele două s-a dovedit a fi rapidă și eficientă. Librăria React a contribuit enorm la realizarea componentei client prin randarea instantă a modificărilor, fapt care a redus semnificativ timpul petrecut dezvoltând această parte. Nu au fost întâmpinate dificultăți nici în ceea ce privește server-ul, dată fiind experiența cu limbajul Python obținută în procesul de realizare al modelului.

5.2 Îmbunătățiri

Atât modelul obținut, cât și aplicația *HeartVitality* , au potențialul de a fi dezvoltate în continuare pentru a îmbunătăți experiența utilizatorului.

Îmbunătățirile aduse modelului depind de scopul urmărit. Dacă dorim să menținem ușurința obținerii datelor, nu există alte atribute relevante care să fie luate în considerare. Am putea atunci să căutăm alți algoritmi de clasificare (modelare sau de tip ansamblu) care nu au fost luați inițial în considerare.

Dacă în schimb, optăm să creștem numărul de atribute din setul de date, indiferent de gradul de dificultate de obținere al acestora, există câteva directii în care

ne putem orienta. Rezultatele unui test ECG, de exemplu, ar putea influența într-un mod semnificativ acuratețea procesului de clasificare. De asemenea, istoricul medical al individului poate fi luat în considerare pentru setul de date, alături de multe boli genetice care pot influența apariția complicațiilor cardiovasculare.

În ceea ce privește aplicația Web, aceasta poate de asemenea fi ulterior îmbunătățită. În momentul actual, site-ul furnizează un rezultat care nu este salvat pentru vizualizarea sa ulterioară. O posibilă îmbunătățire ar putea fi posibilitatea creării unui cont în calitate de pacient. Dacă persoana este conectată cu un asemenea cont, atât datele introduse, cât și rezultatele ar putea fi reținute într-un istoric. Persoana ar avea astfel posibilitatea de a vizualiza evoluția sa în timp. Evident, acest cont nu ar permite accesarea funcționalităților permise administratorilor, așadar conturile ar necesita să aibă un rol/o categorie.

Pentru a putea vizualiza istoricul, dar și completa un nou formular fără a face o tranziție între pagini, pagina principală ar putea, după autentificarea cu succes cu un cont ,,pacient", să permită accesarea unui nou tab care să conțină istoricul.

Site-ul web este momentan gândit pentru uz personal. Dacă s-ar adăuga crearea de conturi, am putea îmbunătăți aplicația astfel încât ea să fie utilizată și în cadrul cabinetelor medicale. Conturile ar putea avea astfel ca posibil rol cel de medic. Contul unui pacient ar putea fi astfel asociat contului medic, astfel încât medicul să poată observa rezultatele și datele pacienților. Medicul ar putea să ofere scheme de tratament pacienților fără un control față-în-față și să fie informat de cazurile grave și să le prioritizeze. Desigur, ar trebui luate măsuri de securitate pentru a garanta protecția datelor personale ale utilizatorului. Pentru a asocia în mod corespunzător rezultatele și datele unui utilizator la contul său, dar și pentru a permite legarea unui cont de pacient la unul de medic ar fi necesară crearea unei baze de date care să conțină tabele pentru conturi, rezultate și asocierile pacient-medic. Dat fiind faptul că nu a fost necesară realizarea de legături între date, o bază de date nu a fost implementată. Pentru contul de medic, tab-ul destinat istoricului pacientului ar putea deveni un tab care să conțină lista sa de pacienți, iar apăsând pe unul dintre ei, medicul să poată vedea rezultatele și datele lor.

Deciziile luate în implementarea aplicației permit dezvoltarea ulterioară a unor funcționalități existente. Dat fiind faptul că frontend-ul a fost dezvoltat utilizând Ionic, codul poate fi reutilizat pentru a dezvolta o aplicație pentru dispozitivele mobile. Modul în care pagina principală poate fi tradusă în română și engleză prin utilizarea fișierelor text poate determina cu ușurință adăugarea de noi limbi. De asemenea pot fi integrați noi algoritmi de clasificare în pagina de administrator.

Bibliografie

[BC] Leo Breiman and Adele Cutler. Random forests. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm/.

[Bro16] Jason Brownlee. *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models and Work Projects End-To-End*. 2016.

[Bur19] Andryi Burkov. *The hundred-page machine learning book*. Andriy Burkov, Quebec City, Canada, 2019.

[dat] datagy. How to Use Python to Calculate Confidence Intervals (3 Methods). https://datagy.io/python-confidence-intervals/.

[DP] DataScience-Prof. Understanding Log Loss: A Comprehensive Guide with Code Examples. https://medium.com/@TheDataScience-ProF/understanding-log-loss-a-comprehensive-guide-with-code-examples

[Fri02] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 2002.

[Gee] GeeksForGeeks. Support Vector Machine Algorithm. https://www.geeksforgeeks.org/support-vector-machine-algorithm/.

[GEW06] Pierre Geurts, D. Ernst, and L. Wegenkel. Extremely randomized trees. *Machine Learning*, 63, 2006.

[GS13] T. V. Geetha and S. Sendhilkumar. *Machine Learning: Concepts, Techniques and Applications*. CRC Press, 2013.

[Har19] Peter Harrington. *Machine Learning in Action*. Manning, Shelter Island, NY, 2019.

[IBM] IBM. What are Naive Bayes classifiers? https://www.ibm.com/topics/naive-bayes.

[Ion] Ionic. Introduction to Ionic. https://ionicframework.com/docs.

[Mara] Regina Maria. Bolile cardiovasculare: categorii, diagnostic, tratament, preventie. https://www.reginamaria.ro/articole-medicale/bolile-cardiovasculare-categorii-diagnostic-tratament-preventie

[Marb] Regina Maria. Bolile cardiovasculare la romani. https://www.reginamaria.ro/articole-medicale/bolile-cardiovasculare-la-romani.

[Org] World Health Organization. Cardiovascular diseases (CVDs). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[Rea] React. Quick Start. https://react.dev/learn.

[Sam59] Arthur L Samuel. Machine learning. *The Technology Review*, 62(1):42–45, 1959.

[sla] scikit learn. GradientBoostingClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[slb] scikit learn. Normalizer. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html.

[TGIH17] Alaa Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien. Linear discriminant analysis: A detailed tutorial. *AI Communications*, 2017.

[Typ] TypeScript. The Basics. https://www.typescriptlang.org/docs/handbook/2/basic-types.html.