Documentatie Algoritm Machine Learning

1. **Introducere**

Scopul unui algoritm de machine learning este sa permita unui sistem informatic sa invete din date si experienta pentru a indeplini o anumita sarcina sau sa faca predictii, fara a fi programat explicit pentru aceasta. In loc sa implementeze reguli sau instructiuni specifice, algoritmii de machine learning utilizeaza modele matematice si statistice pentru a gasi modele și relatii in datele de antrenare și apoi sa generalizeze aceste cunostinte pentru a lua decizii sau a face predictii asupra datelor noi, necunoscute.

Pentru proiectul acesta de practica, unul dintre obiectivele principale a fost realizarea unui astfel de model care sa primeasca mai multe valori de debit de la un aparat si pe baza acestora sa se prezica urmatoarea valoare a debitului pe baza caruia se va calcula costul total al energiei consummate. In realizarea acestui obiectiv au existat mai multe tipuri de modele luate in considerare pentru a implementa alogritmul: regresie liniara, retele neuronale artificiale si arbori de decizie.

Regresia liniara este simplu de invatat si de implementat si este destul de potrivit pentru probleme cu relatii liniare intre variabile, dar nu poate captura corelatii complexe intre acestea, iar performanta poate fi limitata in cazul datelor neliniare.

Arborii de decizie sunt usor de inteles si vizualizat, pot manipula mai multe tipuri de date, dar au tendinta de a fi prea sensibili la mici variatii in datele de intrare.

Retelele neuronale artificiale sunt capabile sa modeleze relatii mai complexe intre variabile si pot invata reprezentari de inalta calitate din date brute, dezavantajele fiind necesitatea unui set de date destul de mare pentru a evita suprainvatarea si faptul ca pot fi dificil de antrenat si de optimizat.

In cazul acestui proiect, algoritmul de machine learning a fost implementat cu ajutorul retelelor neuronale artificiale si se foloseste de un istoric pentru a face predictii. Limbajul de programare folosit este Python, in paradigma orientata obiect, impreuna cu librariile TensorFlow si Keras, ambele specializate in domeniul de machine learning.

1. **Flow-ul principal al aplicatiei**

Pentru a face aceasta aplicatie cat mai prietenoasa cu utilizatorul, am decis sa adaugam un fisier de configuratie (**machine\_learning.config**) in care utilizatorul sa poata completa cu datele pe care acesta le doreste foarte usor, fara a fi nevoie de o cunostinta a codului prea mare. De asemenea, pentru a avea un nivel de transparenta cat mai mare, aplicatia care antreneaza modelul afiseaza output-ul atat in consola cat si intr-un fisier de logging (**logfile.log**). Path-urile catre cele doua fisiere trebuie data ca argumente pentru programul principal care se ocupa cu antrenarea modelului, anume **MainApplication.py** .

Dupa ce modelul a fost antrenat, acesta este salvat in format .onnx si poate fi rulat independent de aplicatia care s-a ocupat cu antrenarea lui, prin rularea programului **Main.py** . Astfel, output-ul acestuia o sa fie o lista cu valorile prezise de algoritmul de machine learning, care urmeaza sa fie preluata mai departe de Node Red si, ulterior, datele incarcate in baza de date din Cloud.

1. **Module folosite**

Printre modulele folosite se numara urmatoarele:

* **os** este utilizat pentru a interactiona cu sistemul de operare sub care ruleaza programul. In aceasta aplicatie este foarte des intalnit pentru a verifica existenta unor fisiere si pentru a le crea, in cazul in care acestea lipsesc (daca este cazul).
* **enum** este folosit pentru a defini tipuri de enumerare, adica pentru a crea un set restrictionat si predefinit de valori constante. Din acest modul am folosit clasa **Enum**, pentru a declara cateva constante, cum ar fi diferite formate si pentru o lista si un dictionar cu care o sa fac verificarea si validarea datelor din fisierul de configuratie.
* **logging** este folosti pentru a inregistra mesaje si informatii utile in timpul rularii programelor. In acest program, logging-ul se face atat in consola cat si in fisierul **logfile.log**.
* **json** este utilizat pentru a lucra cu date in format JSON (Javascript Object Notation), un format de text usor de scris si de citit. Este necesar pentru a parsa datele aflate in fisierul de configuratie care sunt reprezentate in format JSON.
* **argparse** este utilizat pentru a analiza argumentele din linia de comanda atunci cand se ruleaza un script sau o aplicatie in linia de comanda. In aceasta aplicatie, modulul este folosit pentru a oferi ca argumente path-urile catre fisierul de configuratie si catre fisierul de logging.
* **pickle** este un modul folosti pentru serializarea si deserializarea diferitelor obiecte Python intr-un mod usor de stocat si de transmis. In aplicatie este folosit pentru a transmite datele de la o aplicatie la cealalta.
* **pandas** este utilizat in manipularea si analiza datelor intr-un mod eficient si flexibil, oferind structuri de date si functionalitati avansate pentru lucrul cu date tabulare. In aceasta aplicatie este folosit cu precadere pentru a realiza citirea datelor din fisiere de tip csv.
* **numpy** este un modul fundamental in Python, utilizat pentru a lucra cu aranjamente multidimensionale de date si pentru a efectua operatii matematice si statistice avansate. Din acest model, cea mai folosita structura a fost **ndarray** (n-dimensional array) cu care am structurat datele cu care algoritmul de machine learning lucreaza.
* **tensorflow** este un modul open source dezvoltat de Google pentru machine learning si inteligenta artificiala, fiind una dintre cele mai populare si puternice biblioteci pentru construirea si antrenarea modelelor de machine learning, in special modele neuronale. In cadrul acestui modul, am folosit **keras**, o interfata de nivel inalt, care face tot procesul de realizare a unui model cat mai simplu si cu o sintaxa cat mai simplificata.
* **sklearn** este o biblioteca open source specializata in oferirea diferitelor functionalitati care vin in ajutorul dezvoltarii modelelor de machine learning. De aici, in program a fost folosit sub-modulul **model\_selection** pentru a importa functia **train\_test\_split** care faciliteaza impartirea datelor in date pentru antrenament si date pentru testare.
* **tf2onnx** este un modul open source care permite conversia modelelor TensorFlow in formatul ONNX (Open Neural Network Exchange), un format standard pentru reprezentarea retelelor neuronale, care permite interoperabilitatea intre diferite framework-uri de deep learing si biblioteci. De asemenea, aceasta biblioteca poate efectua diferite optimizari in cadrul conversiei pe care o executa.
* **Onnxruntime** este o biblioteca open source dezvoltata de Microsoft pentru rularea modelelor de inteligenta artificiala create in formatul ONNX cu o performanta inalta si eficienta.

1. **Flow-ul principal al aplicatiei**

Aplicatia contine doua programe diferite: **MainApplication.py** si **Main.py.** Primul dintre ele este cel care se ocupa cu antrenarea modelului de machine learning, pe cand ultimul se ocupa cu rulatul acestuia.

**MainApplication.py:** aplicatia incepe prin a parsa argumentele primite in linia de comanda, dupa care initializeaza fisierul logger, validatorul si o instanta a obiectului principal care constituie un „entry-point” pentru acest program. Dupa ce toate aceste obiecte sunt initalizate, se ruleaza functia de executie unde se valideaza urmatoarele: existenta fisierului de configuratie si a unui fisier de logging, corectitudinea datelor din fisierul de configuratie (adica sa nu lipseasca niciun atribut important si sa fie respectat tipul de date al acestora); in cazul in care exista o conditie nerespectata, programul se opreste si arunca o eroare.

Dupa ce datele au fost validate, fisierul de configuratie este parsat si adus in memoria programului. Mai apoi, se initializeaza un obiect **database** care are rolul de a asigura comunicatia cu baza de date din cloud, cu ajutorul careia se face rost de datele de antrenament care sunt aduse local sub forma de fisier in format CSV. Acestea fiind spuse, incepe antrenarea modelului in clasa **MachineLearning**,astfel ca se impart datele in date de antrenament si date de evaluare, dupa care se defineste modelul de retea neuronala, care are diferite functii de activare, se calculeaza eroarea, dupa care se evalueaza performanta lui.

Reteaua neuronala definita este una secventiala, cu cateva straturi diferite. Primul strat contine 64 de unitati, unde este folosita drept funcie de activare **tanh** (tangenta hiperbolica) pentru a gestiona gradientul in timpul antrenarii. Urmatorul strat este un strat complet conectat cu 32 de neuroni care foloseste functia de activare **relu** (rectified linear unit) pentru a ajuta la evitarea gradientului de disparitie si accelerarea convergentei in timpul antrenarii. Stratul al treilea este tot un strat complet conectat cu 16 neuroni care foloseste functia **tanh** de activare. Stratul al patrulea este complet conectat si are 8 neuroni, iar ca functie de activare foloseste **sigmoid**, folosita pentru probleme de clasificare binara sau pentru a obtine valori in intervalul **[0, 1]**. Ultimul nivel este complet conectat si are un singur neuron si are rolul de a produce iesirea directa a retelei.

Dupa ce modelul a fost antrenat si testat, acesta este convertit in format ONNX, iar anumite date sunt salvate in format PKL pentru a putea fi preluate de urmatorul program care se ocupa cu rularea modelului.

**Main.py** incepe prin a prelua datele salvate in formatu PKL si prin a parsa fisierul de configuratie, urmand ca dupa aceea sa incerce sa prezica urmatoarea valoare a debitului. Acesta face 30 de preziceri, prima prezicere bazandu-ze pe ultimele 30 de valori reale ale debitului, urmand ca urmatoarele 29 sa se foloseasca si de valorie deja prezise pentru a face o estimare. Acestea sunt puse intr-o lista si afisate pe ecran, urmand sa fie trimise catre Node Red.

