SFC – Projekt

Genetické algoritmy + fuzzy logika

Jan Zdeněk (xzdene01) November 25, 2024

Contents

1 Úvod do problematiky

Pro projekt do předmětu SFC – Soft Computing bylo zvoleno zadání *GA+FUZZY – aplikace např. v klasifikaci nebo řízení*, tedy zadání číslo 11. V rámci tohoto projektu byl řešen problém predikce jedné výstupní proměnné z hodnot několika vstupních proměnných (úloha pro regresy). Celá implementace proběhla v jazyce python s použitím několika standardních balíčků (numpy, pandas, atd.) a balíčku *skfuzzy*, který je navržen pro práci s fuzzy logikou a nabízí značnou abstrakci nad vnitřní implementací definice fuzzy proměnných a jejich fukcí příslušnosti, fuzzifikace, inference a defuzzifikace.

V rámci vývoje i pro účely ukázky je použit volně dostupný dataset *Concrete Compressive Strength*[1], ketrý obsahuje osm vstupních proměnných (cement, vysokopecní struska, popílek, voda, superplastifikátor, hrubé kamenivo, jemné kamenivo a věk) a jednu výstupní proměnnou (pevnost betonu v tlaku).

2 Genetický algoritmus

Implementace genetického algoritmu je obsažena primárně v souboru *main.py* a funkci `genetic_algorithm()`. Na začátku proběhne inicializace populace a poté N iterací, ve kterých se celá populace ohodnotí a z nejlepšího jedince a mutací je vytvořena následující populace.

2.1 Chromozom

Z pohledu genetického algoritmu je chromozom pouze datová struktura, která reprezentuje daného jedince v populaci. V tomto případě je chromozom, implementován pomocí třídy `Chromosome`,[2] reprezentován několika poli či vektory. Z těchto polí jsou v průběhu tvorbu systému generována pravidla pro fuzzy inferenci. Všechna pole jsou stejné délky, kdy prvky mající totožný index generují jedno pravidlo. Tedy pravidla ve tvaru $P_i = (x_{1i}, x_{2i}, y_i, w_i)$, kde

- $x_{ji} = \overrightarrow{x_j}\,[i]$ (pro $j \in \{1, 2\}$) je j-tý vstup i-tého pravidla;
- $y_i = \overrightarrow{y}\,[i]$ je pravá strana i-tého pravidla;
- $w_i = \overrightarrow{w}\,[i] \in \{0, 1\}$ je váha i-tého pravidla,

, tvoří vektory $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, $\overrightarrow{y}$ a $\overrightarrow{w}$ a dohromady představují jeden chromozom. Co konkrétně chromozom znamená a jak se z něj generují pravidla je popsáno až v podsekci 3.1.

[1] `https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength`

[2] soubor chromosome.py

2.2 Populace

Populaci si lze představit jako množinu chromozomů (jedinců) reprezentující jednu iteraci genetického algoritmu. Populace může být inicializována náhodně pomocí metody `generate_random()` nebo může být načtena ze souboru pomocí přepínače `-i INPUT`.[3] V této implementaci je z populace vybrán vždy jen jeden rodič (nejlepší jedinec podle zvolené metriky) a ten je nakopírován N-krát[4] do nové populace. Jde tedy o strategii $1 + \lambda$, kde jeden rodič generuje všechny potomky.

2.3 Mutace

Posledním krokem pro vygenerování nové populace je mutace každého jedince, která probíhá ve funkci `mutate()`. Vstupem této fáze není jen mutovaný chromozom, ale i 2 koeficienty mutace. První koeficient koresponduje k mutaci vstupů a výstupu pravidla a na druhém je závislá mutace vah (tento rozdíl není dělán jen z důvodu větší kontroly nad optimalizací, ale i z důvodu úplného zakázání mutace vah). V prvním případě je náhodně vygenerováno zcela nové pravidlo a v tom druhém je obrácena hodnota váhy (váhy mohou nabývat pouze hodnot 0 nebo 1).

3 Fuzzy logika

Implementace fuzzy logiky se nachází ve třídě `FuzzySystem`, která však dále využívá i již zmíněnou třídu `Chromosome`. Na začátku optimalizace je tato třída instanciována – k tomuto je potřeba trénovací dataset, aby bylo možné správné definice fuzzy proměnných. Definice proměnných probíhá automaticky, ale s předem stanovenou metodou pro defuzzifikaci a předem určeným krokem pro jednotlivé funkce příslušnosti. V této implementaci je možné zvolit dataset s libovolným počtem vstupních proměnných (ve fuzzy logice pak *antecedent*) a s jednou výstupní proměnnou (ve fuzzy logice pak *consequent*).

3.1 Tvorba pravidel

Chromozom zde reprezentuje zakódovaná pravidla do formy, ve které je možná optimalizace pomocí genetických algoritmů. Jak již bylo zmíněno výše, každé pravidlo je reprezentováno čtveřicí a tedy chromozom obsahuje čtyři pole stejné délky. Pro tvorbu i-tého pravidla je potřeba získat všechny čtyři jemu odpovídající hodnoty.[5] Celé pravidlo je však reprezentováno obecně jako sedmice $P = (a_1, v_1, a_2, v_2, c, v_3, w)$,

[3] `INPUT` reprezentuje cestu k .npz souboru, který obsahuje uložený chromozom

[4] $N = |populace|$

[5] pravidla ve fuzzy logice mohou být i delší, avšak tento fakt byl v tomto projektu ignorován

kde

- a_1 a a_2 jsou antecedenty;
- c je consequent;
- v_1, v_2 a v_3 jsou hodnoty, kterých v pravidlech nabývájí fuzzy proměnné a
- w je váha daného pravidla.

Hodnoty v_i a w jsou pouze jednoduše dekódovány z chromozomu pomocí pole *names*, které mapuje čísla/indexy na již konkrétní jména, která mohou být reprezentována například slovy: *low*, *medium*, *high*. Pole names je tedy polem polí, kde prvek $N_k = names[k]$ je pole všech možných lingvistických hodnot, kterých může k-tá[6] proměnná nabývat.[7] To ke kterým fuzzy proměnným mají být tyto hodnoty přiřazeny je poté zakódováno přímo v pozici kde se pravidlo v poli nachází. Pro lepší porozumění je k dispozici formálnější popis v algoritmu 1.

```
idx = 0
for i in range(len(input_vars)):
    for j in range(len(i+1, input_vars)):
        # zakódováno v pozici
        a1 = in_vars[i]
        a2 = in_vars[j]

        # zakódováno v chromozomu
        v1 = chromosome.a1[idx]
        v2 = chromosome.a2[idx]
        v3 = chromosome.c[idx]
        w = chromosome.w[idx]

        rule[idx] = create_rule(a1, v1, a2, v2, c, v3, w)
        idx += 1
```

Algoritmus 1: Ukázka kódu výše popisuje tvorbu jednotlivých pravidel z chromozomu. Jedná se o těžce zjednodušený příklad, který neodráží skutečnou implementaci. Dále je z ukázky zřejmé, že proměnná c je konstantní, protože existuje jen jeden consequent, a že počet pravidel je pevně určen počtem antecedentů neboli vstupních proměnných.

Tímto procesem je tedy daný chromozom převeden na množinu pravidel, která reprezentuje řídící systém.[8] Z tohoto systému bude později tvořena simulace, která pracuje již s konkrétními hodnotami.

[6] pozor jedná se o k-tou proměnnou nikoli pravidlo, k je dekódováno z pozice v chromozomu viz. algoritmus 1

[7] příklad: $a_i = teplota$, $v_i = 2$, $N_{teplota} = [low, medium, high]$, pak bude tento antecedent v pravidle vypadat: $teplota[medium]$

[8] `ControlSystem` v balíčku *skfuzzy*

3.2 Simulace

Simulací je rozuměna transformace vstupních proměnných na výstupní proměnnou aplikováním předem definovaných pravidel. Pravidla, a tedy i kontrolní systém, jsou generována pro každý chromozom – ten představuje unikátní množinu pravidel. Na rozdíl od toho je simulace tvořena pro každý záznam z datasetu – simulace již obsahuje konkrétní hodnoty proměnných. Tento proces lze popsat v několika krocích:

1. **vytvoření** objektu **simulace** z kontrolního systému (pravidel);
2. **vložení hodnot** pro každou vstupní proměnnou;
3. **fuzzifikace** vstupních proměnných;
4. **fuzzy inference** neboli aplikace jednotlivých pravidel a
5. následná **defuzzifikace** výstupní hodnoty, tedy převedení na již konkrétní číslo.

Kroky 3, 4 a 5 probíhají zcela automaticky s pomocí python knihovny *skfuzzy* a tedy zde nejsou popsány.

4 Návod

Pro instalaci je nutné pouze zreplikovat prostředí v nástroji *conda* a to příkazem

```
conda env create --file environment.yaml --name sfc
```

, kdy po jeho aktivaci je již možné spustit *main.py* pro optimalizaci nebo *test.py* pro evaluaci daného chromozomu. Jak spouštět jednotlivé skripty je popsané v přiloženém souboru *README.md* nebo lze při spuštění zvolit přepínač `-h`.

4.1 Optimalizace

Před započetím optimalizace je potřeba zvolit dataset, parametry pro genetický algoritmus, metriku pro ohodnocení jedince a obecné parametry pro běh programu. Všechny tyto parametry mají přidělenou výchozí hodnotu a tedy je možné skript *main.py* spustit i bez jejich určení, ale i přesto jsou zde některé vysvětleny:

- **cesta k datasetu** – dataset musí být ve formátu .csv a musí obsahovat právě jeden sloupec, ketrý reprezentuje predikovanou hodnotu (v ukázkovém případě se jedná o pevnost betonu v tlaku, angl. *Concrete compressive strength*) a nese název *target*;
- většina vstupních parametrů je typická pro tuto metodu optimalizace, avšak parametry **active_rules**, tedy poměr pravidel s váhou 1 při inicializaci, a **a_mutation**, tedy koeficien mutace pro váhy pravidel, stojí za zmínku;
- **přepínač pro testování** poté určuje zda má být provedena ještě výsledná evaluace celého systému na datasetu a

- v poslední řadě lze zvolit **vstupní chromozom**, který bude použit pro inicializaci populace místo náhodného generování pravidel.

Výsledný chromozom a konfigurace pro jeho získání jsou po provedení optimalizace uloženy do složek chromosomes/ a configs/. Názvy těchto souborů jsou ukončeny časovou značkou pro zamezení přepisování již předtím uložených souborů. Mimo chromozomu je pro predikci potřebné mít uložená i metadata o systému, aby bylo možné správně sestavit pravidla z chromozomu. Tato metadata jsou uložena do souboru *system.json*, ale pozor tento soubor se po každém spuštění přepisuje!

4.2 Testování

Při testování chromozomu je potřeba poskytnout pouze dataset, chromozom a metadata systému. Výsledkem je poté mírně detailnější analýza správnosti chromozomu, která je vypsána na standardní výstup programu. Například pro vypsání výsledků ukázkového chromozomu je možné skript spustit pomocí příkazu:

```
python test.py -c chromosomes/chrom.npz
```

. Výsledky ukázkového chromozomu jsou i uloženy v souboru *result.txt*.