FLP - 1. projekt (rozhodovací stromy)

Rozhodovací stromy jsou stromové struktury používané v rozhodování a predikci. Jsou složeny z uzlů a hran, kde každý uzel reprezentuje rozhodnutí na základě určitého atributu, a každá hrana odpovídá jedné z možností tohoto rozhodnutí. Listy stromu reprezentují konečné výsledky nebo třídy, které se snažíme předpovědět nebo klasifikovat. Tyto stromy se používají v různých oblastech, jako jsou strojové učení, datová analýza a rozhodovací podpora. Jsou populární pro svou schopnost poskytnout srozumitelný a interpretabilní model, který je vhodný pro lidské pochopení.

Program funguje ve dvou režimech: klasifikace a trénování rozhodovacího stromu.

- Klasifikační režim: V tomto režimu program přijímá vstupní data a klasifikuje je na základě předem vytvořeného rozhodovacího stromu.
- Trénovací režim: V tomto režimu program přijímá vstupní data a používá je k trénování nového rozhodovacího stromu.

Co složka obsahuje

Ve složce s projektem najdeme:

- Makefile - Soubor sloužící pro překlad programu.
- proj.hs - Hlavní soubor obsahující počáteční logiku.
- dataStructures.hs - Soubor s datovými strukturami.
- parseFile.hs - Soubor s logikou parsování všech možných souborů.
- trainTree.hs - Soubor s logikou trénování nového stromu.
- doc.pdf - Toto pdf.

Kompilace a spuštění

Program lze zkompilovat příkazem *make*, který ze zdrojových souborů vytvoří spustitelný program.
Příkaz *make clean* pak smaže veškeré objektové a binární soubory.

Pro spuštění klasifikačního režimu použijte příkaz: *./flp-fun -1 <soubor se stromem> <soubor s daty>*
Pro spuštění trénovacího režimu použijte příkaz: *./flp-fun -2 <soubor s trenovacimi daty>*
Pokud zadáte špatné argumenty, program vypíše nápovědu.

Popis fungování algoritmů

Klasifikace

Proces klasifikace začíná funkcí `classifyMode` v souboru `proj.hs`. Nejprve je potřeba zpracovat soubor a vytvořit strom podle specifikace. To se děje ve funkci `parseTree` uvnitř souboru `parseFile.hs`. Pokud se vyskytne chyba (například strom je nesprávně zapsán),

chyba je vypsána do terminálu. Zde jsou některé zdroje, které byly využity při učení se zpracování souborů:

- https://book.realworldhaskell.org/read/using-parsec.html
- https://riptutorial.com/haskell/example/4413/using-implication-------to-check-properties-with-preconditions
- https://hackage.haskell.org/package/CheatSheet-1.11/src/CheatSheet.pdf

Dále je potřeba zpracovat soubor s daty a každé dato provést stromem, aby byla zjištěna výsledná třída. To probíhá ve funkci `treeFind`. Pro pochopení, jak na zpracování souboru s daty řádek po řádku, kde na každý řádek je aplikována nějaká logika, bylo využito známé knihy:

- https://learnyouahaskell.com/input-and-output

Výstupem tohoto modulu je výpis všech tříd po řádcích tak, jak jsou zapsány v souboru.

Trénování

Nejprve je zapotřebí dataset zpracovat a uložit do vnitřní struktury. Zpracování probíhá tak, že všechna data na řádku až na poslední jsou brána jako čísla, dle kterých se strom vypočítává. Poslední číslo (nebo řetězec) je brán jako označení třídy. Tedy čtení probíhá řádek po řádku, vezme se záznam,zjistí se, zda-li je za ním čárka, a pokud ano, je uložen jako double hodnota. Jinak je to poslední záznam na řádku a proto je uložen jako třída.

V souboru `trainTree.hs` probíhá trénování nad zpracovanými daty. Algoritmus byl pochopen na základě tohoto videa: https://www.youtube.com/watch?v=_L39rN6gz7Y, a tak byl i implementován.
Na vstup funkce `buildTree` jde dataset, který sestává ze seznamu řádků, kde jeden řádek je seznam možných hodnot, kde tyto hodnoty jsou buďto `Numeric Double`, nebo `Label String`. Na konci vrací vypočítaný rozhodovací strom.

Určování rozhodovacího stromu začíná rozdělením datové sady na dvě poloviny na základě hodnoty aktuálního uzlu. Tento proces je prováděn rekurzivně, přičemž jsou vypočítávány jednotlivé hodnoty Gini impurity. Tyto hodnoty určují, který sloupec a s jakou hodnotou se má vytvořit jako další uzel ve stromu.
Pro fungování algoritmu nahlédněte do zdrojového souboru, byla snaha o kvalitní dokumentaci.

Testování

Testování probíhalo na autorově vytvořených variancích i za pomoci referenčních testů.
Zde jsou výstřižky testů z clusteru Merlin.

Klasifikace:

```
xjahnf00@merlin: ~/FLP/public$
xjahnf00@merlin: ~/FLP/public$ python3 test_flp.py --test_type inference
Running inference tests: 100%|                                                                                | 4/4 [00:00<00:00, 18.74it/s]
All inference tests PASSED (4/4).
xjahnf00@merlin: ~/FLP/public$
```

Trénování:

```
xjahnf00@merlin: ~/FLP/public$
xjahnf00@merlin: ~/FLP/public$ python3 test_flp.py --test_type training
Running training tests: 100%|                                                                                | 12/12 [00:21<00:00,  1.83s/it]
xjahnf00@merlin: ~/FLP/public$
xjahnf00@merlin: ~/FLP/public$ cat training_tests_result.csv
dataset,split,train_accuracy,test_accuracy,reference_train_accuracy,reference_test_accuracy,test_difference,train_time
penguins_all.csv,0, 1.000, 1.000, 1.000, 1.000, 0.000, 0.209
penguins_all.csv,0.25, 1.000, 0.976, 1.000, 0.976, 0.000, 0.251
penguins_all.csv,0.5, 1.000, 0.910, 1.000, 0.958, 0.048, 0.119
iris_all.csv,0, 1.000, 1.000, 1.000, 1.000, 0.000, 0.073
iris_all.csv,0.25, 1.000, 1.000, 1.000, 1.000, 0.000, 0.085
iris_all.csv,0.5, 1.000, 1.000, 1.000, 1.000, 0.000, 0.044
housing_all.csv,0, 0.975, 0.975, 1.000, 1.000, 0.025, 1.383
housing_all.csv,0.25, 0.851, 0.586, 1.000, 0.646, 0.061, 0.707
housing_all.csv,0.5, 1.000, 0.670, 1.000, 0.640,-0.030, 0.326
wines_all.csv,0, 0.875, 0.875, 1.000, 1.000, 0.125, 9.429
wines_all.csv,0.25, 0.664, 0.472, 1.000, 0.577, 0.105, 5.099
wines_all.csv,0.5, 0.709, 0.416, 1.000, 0.531, 0.115, 2.537
xjahnf00@merlin: ~/FLP/public$
```