[PSZT] Uczenie maszynowe – Projekt 2.
Wojciech Wolny
Grzegorz Rypeść

1. Treść projektu
Przewidywanie czy grzyb jest jadalny przy użyciu własnej implementacji algorytmu ID3. Testy binarne.

2. Oczekiwania od naszego modelu
Algorytm CART zaimplementowany w bibliotece scikit-learn uzyskał dokładność 100% na analizowanym zbiorze danych przy użyciu walidacji krzyżowej dla k=5. Test ten znajduje się w pliku tree.py.
Algorytm id3 zaimplementowany w bibliotece decision-tree-id3 uzyskał dokładność 100% na analizowanym zbiorze danych przy użyciu walidacji krzyżowej dla k=5. Test ten znajduje się w pliku depth_analysis.py.
Oczekujemy zatem, że drzewo decyzyjne, stworzone przez nasz algorytm uzyska podobne wyniki.

3. Zbiór danych
Dane pochodzą ze strony https://archive.ics.uci.edu/ml/datasets/mushroom
Na potrzeby zadania plik został przekonwertowany do formatu csv, dodane zostały nagłówki do kolumn, kolumnę klas przesunęliśmy na koniec, a dane zostały ustawione w kolejności losowej.

Każdy z 8124 wektorów danych składa się z 22 kategorycznych atrybutów oraz 1 klasy (jadalny/niejadalny). Atrybut “stalk-root” zawiera brakujące pola, to jest 2480 wierszy, w których wartość atrybutu wynosi “?”. W toku testowania naszego drzewa atrybut ten nie był brany pod uwagę w czasie budowy drzewa, stąd nie mogliśmy oszacować czy lepiej pozostawić kolumnę czy ją usunąć. Uznaliśmy jednak, że atrybut ten niesie informacje i pozostawiliśmy “?” jako dodatkową wartość atrybutu “stalk-root”.

4. Wykorzystane narzędzia i biblioteki
- Python 3.6
- Atom, PyCharm – edytor tektu oraz IDE
- Pandas – biblioteka do operowania na danych tablicowych/macierzach
- Numpy – biblioteka do obliczeń tensorowych
- git – system kontroli wersji
- github - https://github.com/grypesc/id3Tree
- decision-tree-id3 – wykorzystany algorytm ID3 do analizy porównawczej
- sklearn – narzędzia i drzewo CART
- matplotlib, seaborn – biblioteki do wizualizacji danych

5. Plik tree.py
Plik ten implementuje klasę DecisionTreeClassifier zwierającą klasę zagnieżdzoną Node. DecisionTreeClassifier udostępnia następujące publiczne metody:
- fit(X, Y, max_depth) – Buduje drzewo decyzyjne dopasowując je do danych
- predict(X) – Zwraca wektor Y powstały przez klasyfikację danych z macierzy X.
- evaluate(X, Y) – Zwraca dokładność predykcji na macierzy X
- print_tree(node) – Zwraca wszystkie węzły decyzyjne i ich głębokość

Metody prywatne to:

- __id3__(node) – Rekurencyjnie wywoływana metoda, która dla danego węzła wybiera najlepszy atrybut i na jego podstawie tworzy dzieci. Bazuje na algorytmie podanym na wykładzie i posiada podobne oznaczenia.
- __infGain__(X, Y) – Metoda statyczna, zwraca information gain dla macierzy X i wektora Y

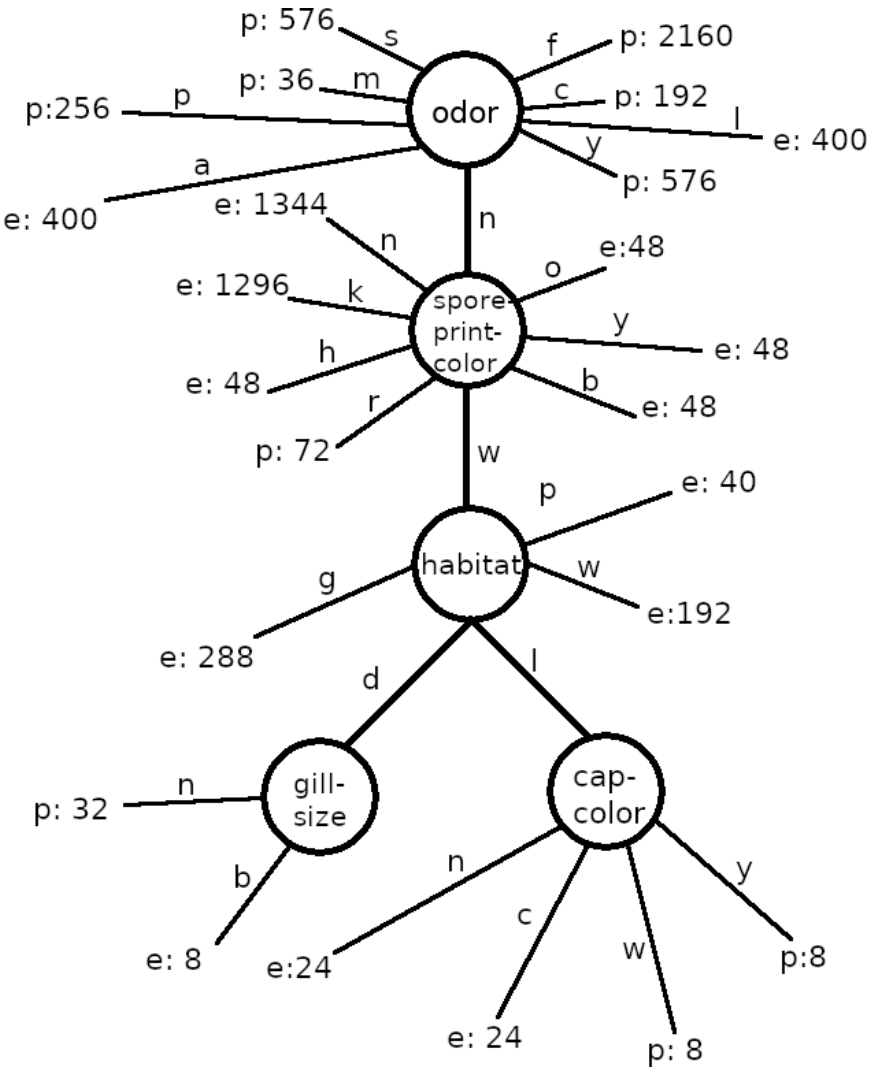
6. Podział na role

Grzegorz – Stworzenie pliku tree.py, implementacja klas Node i DecisionTreeClassifier, pisanie dokumentacji, analiza powstałego drzewa decyzyjnego
Wojciech – Implementacja i przygotowanie programów testowych, analiza wyników, pisanie dokumentacji

7. Problemy

W toku testowania algorytmu pojawił się problem, gdy drzewo miało dokonać predykcji punktu, którego wartość atrybutu nie wystąpiła w zbiorze trenującym. Nie mogliśmy znaleźć w materiałach do przedmiotu odpowiedzi na pytanie co zrobić w takiej sytuacji, więc uznaliśmy, że taki punkt zostanie zaklasyfikowany do klasy, która dominuje w węźle, w którym pojawił się ten problem.

8. Wyniki dla nieograniczonej głebokości



Ilustracja 0: Uzyskane przez nas drzewo decyzyjne dla całego zbioru danych „data.csv”. Liczba w liściach oznacza liczbę obiektów zaklasyfikowanych jako ‘poisonous’, lub ‘edible’.

Aby sprawdzić czy drzewo zostało poprawnie utworzone wypisaliśmy je i zobrazowaliśmy na ilustracji 0. Następnie dla 20 punktów ręcznie sprawdzaliśmy czy podążając ścieżką wg wartości atrybutów dojdziemy do liścia z odpowiednią klasą. Wszystko jest w porządku.

Nastepnie porównaliśmy osiągane wyniki z drzewem CART z biblioteki scikit-learn używając 5-krotnej walidacji krzyżowej. Obydwa algorytmy uzyskały dokładność 100%, natomiast dla maksymalnej głebokości równej 3 osiągnęły 98%, co jest zgodne z oczekiwaniami. Dokładność drzewa zmniejsza się wraz z mniejszą liczbą węzłów decyzyjnych. Aby odtworzyć te wyniki należy wykonać plik tree.py.

9. Porównanie dwóch implementacji drzewa id3

W celu ponownego otrzymania wyników analizy porównawczej dla dwóch implementacji algorytmu drzewa ID3 należy uruchomić program depth_analysis.py. W celu odtworzenia wyników pojedynczej analizy skuteczności dla drzewa należy uruchomić program tree.py. Pierwsza analiza, którą przeprowadzilismy polegała na ocenie dokładności naszego algorytmu dla podanych danych. Dla stworzonego algorytmu przygotowalismy testy w celu wykrycia dokładności algorytmu dla różnych głębokości i różnej wielkości danych treningowych. Dla pierwszych kilku testów otrzymywaliśmy bardzo wysoki wynik skuteczności, dlatego zdecydowaliśmy porównać maksymalne głębokości 3 i 4 oraz dla każdej maksymalnej głębokości od 10% do 80% włącznie danych przeznaczonych do treningu. W wyniku naszego testu wyszło, że dla wszystkich zaproponowanych wielkości danych treningowych drzewo wybiera odpowiednie atrybuty do drzewa i osiąga skuteczność na poziomie 99% dla głębokości równej 3 i wybiera atrybuty zapach oraz kolor zarodnika grzyba, a dla głębokości 4 osiągneliśmy skuteczność 100% przy dodatkowym wykorzystaniu atrybutu środowisko, w którym wyrasta grzyb.

```
depth 3 train 10 percent: 0.99 accuracy
odor 1
spore-print-color 2
```

Ilustacja 1. Wyniki dla głębokości 3 i przeznaczeinu 10% na dane treningowe.

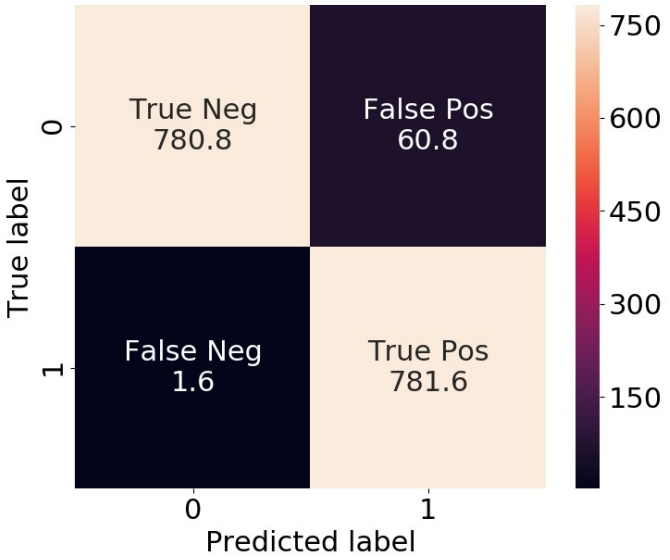
```
depth 4 train 10 percent: 0.99 accuracy
odor 1
spore-print-color 2
depth 4 train 20 percent: 1.00 accuracy
odor 1
spore-print-color 2
habitat 3
```

Ilustracja 2. Wyniki dla głębokości 3 i 80% danych treningowych oraz 4 i 10% danych treningowych.

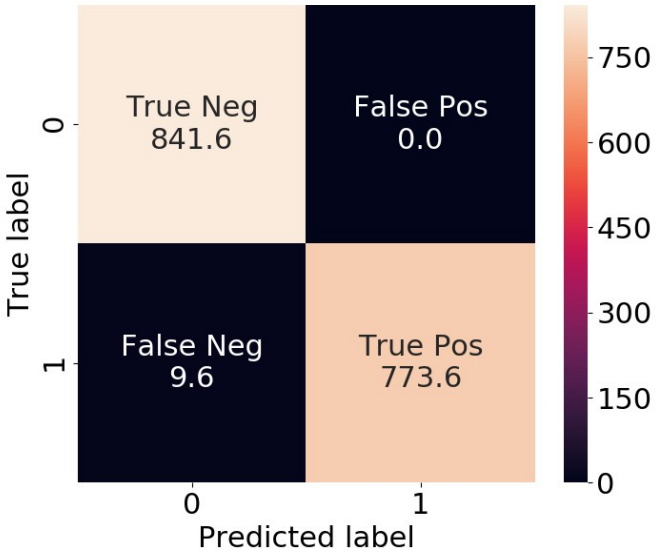
Jak widać na powyższych logach nasze drzewo osiąga bardzo wysoką dokładność dla małych głębokości. Co więcej, przy trenowaniu na wszystkich 100% punktów wykorzystywane jest tylko 5 atrybutów: odor, spore-print-color, habitat, gill-size, cap-color. Możemy stąd wnioskować, że to właśnie one są wystarczające do klasyfikowania jadalności grzybów, a reszta atrybutów w tym zbiorze jest zbyteczna dla problemu klasyfikacji. Dokładność naszego drzewa rośnie wraz z jego głębokością lub zwiększeniem zbioru treningowego, co jest zgodne z oczekiwaniami.

Druga analiza opierała się na porównaniu naszej implementacji drzewa ID3 oraz ogólnodostępnej implementacji. Wybraliśmy algorytm z biblioteki decision-tree-id3 w

wersji 0.1.2 kompatybilnej z biblioteką scikit-learn stworzony przez Daniela Petterssona. Wybraliśmy metodę k krotnej walidacji krzyżowej w celu dokonania porównania. Zdecydowaliśmy się ustawić masymalną głębokość na poziomie 3 oraz przyjęliśmy k równe 5. Analizę zaczęliśmy od przygotowania danych w pliku prepare_data.ipynb, gdzie przemieszalismy dane. Następnie w pliku k_fold_cross_vallidation.py zaczęliśmy od podzielenia danych na K zbiorów. Potem dla każdego zbioru, będącego zbiorem testowym, wybieraliśmy resztę zbiorów jako zbiór treningowy. Dzięki takiemu rozwiązaniu mogliśmy porównać skuteczność naszego drzewa dla różnych zbiorów danych treningowych i walidacyjnych. Po obliczeniu sumy wyników, w których model dawał wyniki prawdziwie pozytywne, prawdziwie negatywne, fałszywie pozytywne i fałszywie negatywne. Każdą tę liczbę podzieliliśmy przez wartość K i stworzyliśmy macierze błędów dla obu implementacji. Na poniższych ilustacjach widać, że nasza implementacja okazała się lepsza dla wyników fałszywie pozytywnych. Dla osoby, która chciałaby skorzystać z naszego algorytmu w celu wykrycia, czy znaleziony grzyb jest jadalny, informacja ta mówi jaki jadalny grzyb zostałby przez algorytm oznaczony jako trujący. Niestety w dużo ważniejszej kategorii w przypadku grzybów wyników fałszywie negatywnych nasz algorytm dał nieznacznie gorsze wyniki. Kategoria ta mówi kiedy algorytm myli się co do tego czy grzyb jest trujący. W tym wypadku oznacza to, że jest większa szansa na to że grzyb, który według naszego drzewa ID3 będzie jadalny okaże się trujący. Ze względu na bezpieczeństwo grzybiarza lepiej, by algorytm miał większą czułość niż precyzję.



Ilustacja 3. Wyniki walidacji k-krotnej krzyżowej dla algorytmu ID3 zaimplementowanego przez Daniela Petterssona.



Ilustacja 4. Macierz błędu dla naszego algorytmu drzewa ID3.

Jednak dla nieograniczonej głębokości obydwa algorytmy nie mylą się ani razu osiągając:

- dokładność = 1.00
- precyzja = 1.00
- czułość = 1.00

Parametry te są liczone poprzez uśrednienie wyników walidacji krzyżowej, nasz ostateczny, najlepszy model został nauczony na całym zbiorze danych.