



Politechnika Krakowska
Wydział inżynierii Elektrycznej i Komputerowej

Studia Niestacjonarne, Informatyka w Inżynierii Komputerowej
Sprawozdanie z projektu z przedmiotu:

Przetwarzanie rozproszone i równoległe

Temat:

**Las losowy. Testowanie szybkości działania
predykcji dla własnej implementacji w
środowisku PVM.**

**Wykonali:
Konrad Groń
Mateusz Rysula
Damian Golonka**

Spis treści

PROBLEM	3
Wstęp ogólny	3
Wstęp teoretyczny	3
Struktura drzewa	4
Nasza implementacja	4
Jak to działa naprawdę ?	4
Entropia informacji	5
Zysk informacyjny	6
Algorytm ID3	6
Implementacja	7
Testy	7
Kod rozwiązania opis ogólny	8
Rozwiązywany problem	9
Opis zbioru danych	9
Procedura pozyskania nowych danych	11
Przykład realizacji drzewa decyzyjnego	11
Dane testowe	11
Kod testujący rozwiązanie	12
Dane wyjściowe	13
Skuteczność modelu	18
Macierz pomyłek	18
Dygresja	18
Czas	19
Lasy losowe	20
Model zespołowy	20
Kod rozwiązania	20
Opis głównej funkcji	20
Testy rozwiązania	21
Wyniki	21
Finalne predykcje	27
Czas	28
Złożoność obliczeniowa	28
Wady i zalety algorytmu ID3	28
Eksperymenty	29
Uczenie lasu z 15 wierszy danych uczących na każde drzewo	29
Czasy budowy i predykcji	29

Wykres czasu predykcji i skuteczności modelu	29
Uczenie lasu z 80 wierszy na każde drzewo	30
Wykres skuteczności zależnie od ilości drzew w lesie.	30
Wykres czasu obliczeń w stosunku do ilości drzew	31
Podsumowanie	31
Cześć 2 Środowisko PVM.....	32
Opis środowiska PVM	32
Architektura rozwiązania	32
Kod rozwiązania.....	33
Przykład uruchomienia kodu.....	37
Szybkość działania	38
Maszyny wirtualne	38
Testy na fizycznych komputerach	40
Wykresy	40
Wnioski	40
Kolejne kroki	41

PROBLEM

W ramach naszego projektu chcieliśmy zaimplementować kod rozwiązania które rozpoznaje gatunki kwiatów po pomiarach jego długości, szerokości płatków oraz długości, szerokości kielicha. W tym celu wybraliśmy że zaimplementujemy model lasów losowych.

Wstęp ogólny

W celu realizacji rozpoznawania gatunku kwiatów zdecydowaliśmy się stworzyć od zera bez użycia gotowych bibliotek model lasu losowego czyli jeden z modeli standardowego uczenia maszynowego. Ten model nie został wybrany do danych jak to ma zazwyczaj miejsce a odwrotnie ponieważ chcieliśmy zobaczyć jak pod spodem działa taki model dlatego zdecydowaliśmy się go użyć jednakże dla tego typu danych ten model sprawuje się mimo wszystko dobrze więc był to tak czy inaczej trafiony wybór.

Wstęp teoretyczny

Drzewo decyzyjne to nic innego jak graficzny sposób wspierania procesu decyzyjnego. Drzewo stosowane jest w teorii decyzji i ma sporo zastosowań. Może zarówno rozwiązać problem decyzyjny, jak i stworzyć plan. Metoda drzew decyzyjnych sprawdza się przede wszystkim, kiedy mamy problemy decyzyjne z wieloma rozgałęziającymi się wariantami oraz kiedy podejmujemy decyzję w warunkach ryzyka.

Struktura drzewa

Drzewem decyzyjnym jest to graf-drzewo, które składa się z korzenia, węzłów, krawędzi oraz liści. Liście to węzły, z których nie wychodzą już żadne krawędzie. Korzeń drzewa tworzony jest przez wybrany atrybut, natomiast poszczególne gałęzie reprezentują wartości tego atrybutu. Dzięki drzewu decyzyjnemu, zbudowanemu na podstawie danych empirycznych, można sklasyfikować nowe obiekty, które nie brały udziału w procesie tworzenia drzewa. Drzewa decyzyjne charakteryzują się strukturą hierarchiczną. Znaczy to, że w kolejnych krokach dzieli się zbiór obiektów, poprzez odpowiedzi na pytania o wartości wybranych cech lub ich kombinacji liniowych. Ostateczna decyzja zależy od odpowiedzi na wszystkie pytania. W algorytmach konstrukcji drzew jednym z kluczowych elementów jest wybór kolejności cech, według których, na poszczególnych etapach, będzie dokonywany podział zbioru obiektów. Technika drzew decyzyjnych to uzupełnienie metod klasycznych. Przykładem może tu być analiza dyskryminacyjna. Hierarchiczność podejmowania decyzji jest cechą, która wyróżnia drzewo decyzyjne od innych metod.

Nasza implementacja

Przy opisie i realizacji przez nas algorytmu lasu losowego używany jest tylko podzbiór możliwych realizacji jego budowy dla przykładu przy obliczaniu zysku informacyjnego używamy entropii informacji natomiast np. w bibliotece scikit-learn używa w tym celu wskaźnika Giniego. Moduł scikit-learn używa również algorytmu CART do generowania drzew, buduje on tylko drzewa binarne. My użyliśmy jednak w naszej implementacji algorytmu ID3. Nasz model nie zawiera też narzędzi regularyzacji i wielu innych rozwiązań które są obecne w profesjonalnych implementacjach tego algorytmu.

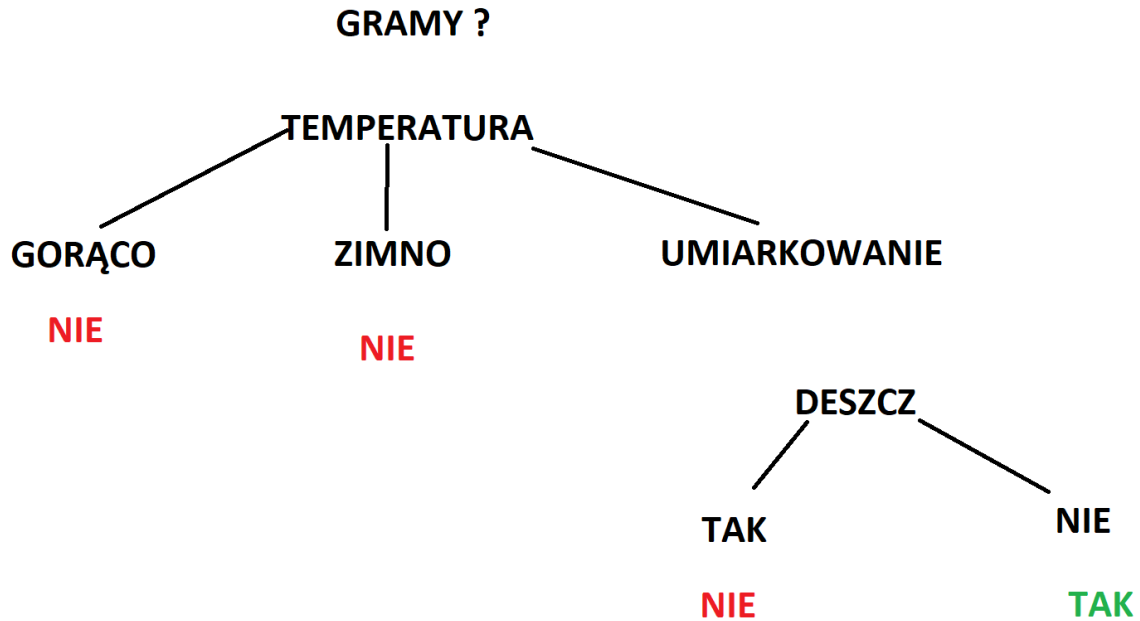
Jak to działa naprawdę ?

Założmy że chcemy zagrać w piłkę nożną. Czasami gdy warunki pogodowe są niesprzyjające jednak nie mamy na to ochoty. Spróbujmy stworzyć teoretyczny model który za nas podejmie decyzję czy mamy na to ochotę. Najpierw zebraliśmy nasze dane z poprzednich umówień się na piłkę i z tym czy faktycznie graliśmy czy nie.

Temperatura	Czy pada ?	Gramy?
Zimno	Nie	Nie
Zimno	Tak	Nie
Umiarkowanie	Nie	Tak
Umiarkowanie	Tak	Nie
Gorąco	Nie	Nie
Gorąco	Tak	Nie

Jesteśmy wybredni, musi być umiarkowanie gorąco i nie padać abyśmy rozegrali mecz. Chyba większość czasu siedzimy na kanapie 😊.

Spróbujmy zwizualizować to w formie drzewa



Jak możemy zobaczyć warunkiem koniecznym do gry w piłkę jest temperatura umiarkowana. Pytanie o to czy pada jest zasadne tylko wtedy gdy wiemy jaka jest pogoda. Nasz proces podejmowania decyzji jest więc hierarchiczną strukturą którą możemy zwizualizować jako drzewo. Liście w tej strukturze reprezentują naszą decyzję. Podejmując decyzję przechodzimy od korzenia w dół struktury w kolejnych węzłach podejmujemy decyzje gdzie przejść na podstawie wartości atrybutu na danym poziomie. Koniec naszej drogi jest w węźle który warunkuje nam decyzje.

Entropia informacji

Entropia – średnia ilość informacji, przypadająca na pojedynczą wiadomość ze źródła informacji. Innymi słowy jest to średnia ważona ilości informacji niesionej przez pojedynczą wiadomość, gdzie wagami są prawdopodobieństwa nadania poszczególnych wiadomości.

Wzór

$$H(X) = \sum_{i=1}^n p(x_i) \log_r \frac{1}{p(x_i)} = - \sum_{i=1}^n p(x_i) \log_r p(x_i),$$

Dla przykładu dla rzutu symetryczną monetą mamy

$$E = -0.5 * \log_2(0.5) - 0.5 * \log_2(0.5) = 1$$

Dla monety w której reszka wypada 3 razy częściej od orła mamy

$$E = -0.25 * \log_2(0.25) - 0.75 * \log_2(0.75) = 0.811278$$

Poniżej jeden co oznacza że teoretycznie moglibyśmy skompresować wiadomość zawierającą 75 procent bitów zero i 25% bitów jeden. Do rozmiaru około 81,12% wiadomości początkowej kodowanej za pomocą bitów 0 i 1.

Zysk informacyjny

Jest to redukcja entropii przy wykorzystaniu danego atrybutu.

$$IG(A) = I - \text{Entropia warunkowa}(A)$$

Prościej mówiąc jest to wielkość o jaką zwiększy się entropia zbioru wskutek zastosowania pewnej procedury mającej wpływ na prawdopodobieństwo wystąpienia elementów. Na przykład sytuacja kiedy rzucamy na raz 3 monetami. Wtedy entropia wynosi 3 bo mamy 8 tak samo prawdopodobnych rezultatów, natomiast kiedy rzucimy trzema ale zobaczymy na jedna bez pozostałych dwóch to entropia wyniesie 2 a możliwe rezultaty będą 4. Patrzenie na monetę jest więc procedurą dającą zysk informacyjny wynoszący jeden.

Na podstawie naszych danych policzmy zysk informacyjny dla obu naszych kolumn. Na początku jednak musimy policzyć entropie atrybutu decyzyjnego Gramy. Tylko w jednym na 6 wierszy występuje wartość tak więc entropia wynosi.

$$E(S) = -1/6 * \log_2(1/6) - 5/6 * \log_2(5/6) = 0.6500$$

Teraz policzmy wartość entropii dla kolumny temperatura.

Mamy 3 wartości tego atrybutu (Zimno, Umiarkowanie, Ciepło) liczymy więc dla poszczególnych podzbiorów i tak

ZIMNO

$$E(S \text{ ZIMNO}) = -2/2 * \log_2(2/2) = 0$$

ponieważ obie wartości w kolumnie decyzji to nie

$$E(S \text{ GORĄCO}) = 0$$

z tego samego powodu obliczenia jak wyżej

$$E(S \text{ UMIARKOWANIE}) = -1/2 * \log_2(1/2) - 1/2 * \log_2(1/2) = 1$$

Wobec tego zysk informacyjny to

$$IG(\text{Temperatura}) = E(S) - (2/6 * E(\text{Zimno}) + 2/6 * E(\text{Umiarkowanie}) + 2/6 * E(\text{Ciepło})) = 0.650022 - 1/3 = 0.3166890$$

Gdybyśmy wybrali deszcz jako kryterium podziału otrzymalibyśmy dwa podzbiory odpowiednio o entropii

$$E(\text{Tak}) = 0$$

ponieważ wszędzie jest nie

Z kolei

$$E(\text{Nie}) = -2/3 * \log_2(2/3) - 1/3 * \log_2(1/3) = 0.9182$$

Wynikowo zysk informacyjny to (analogicznie liczone jak wyżej)

$$IG(\text{Deszcz}) = 0.19087$$

Algorytm ID3

Budowanie drzewa rozpoczynamy od korzenia reprezentującego cały zbiór. Z korzenia wyprowadzamy gałęzie odpowiadające podziałowi zbioru każda z nich odpowiada jednej

wartości atrybutu wybranego jako kryterium podziału. Budowę całego drzewa opisuje poniższy algorytm

Opis algorytmu

1. Drzewo zaczyna od pojedynczego węzła reprezentującego cały zbiór treningowy.
2. Jeżeli wszystkie przykłady należą do jednej klasy decyzyjnej, to zbadany węzeł staje się liściem i jest on etykietowany tą decyzją.
3. W przeciwnym przypadku algorytm wykorzystuje miarę entropii (funkcja przyrostu informacji) jako heurystyki do wyboru atrybutu, który najlepiej dzieli zbiór przykładów treningowych.
4. Dla każdego wyniku testu tworzy się jedno odgałęzienie i przykłady treningowe są odpowiednio rozdzielone do nowych węzłów (poddrzew).
5. Algorytm działa dalej w rekurencyjny sposób dla zbiorów przykładów przydzielonych do poddrzew.
6. Algorytm kończy się, gdy kryterium stopu jest spełnione.

Klasyfikacja przykładu

Gdy już zbudujemy drzewo decyzyjne dla zbioru danych, na podstawie atrybutu X , jego elementów i klas decyzyjnych możemy drzewo to wykorzystywać do klasyfikowania owych elementów danych, opierając się na wartościach ich atrybutów. Dla badanego elementu, znając wartość atrybutu A , pełniącego rolę klasyfikatora w korzeniu drzewa decyzyjnego, wyruszamy z korzenia wzdłuż gałęzi odpowiadającej tej wartości. W węźle, do którego dotarliśmy, powtarzamy tę czynność (oczywiście biorąc pod uwagę atrybut związany z tym węźlem) i tak dalej, dla kolejnych węzłów, aż dotrzemy do liścia. Ów liść reprezentować będzie atrybut decyzyjny, czyli jedną z klas rezultatu, do której ostatecznie zaliczony zostanie badany element.

Implementacja

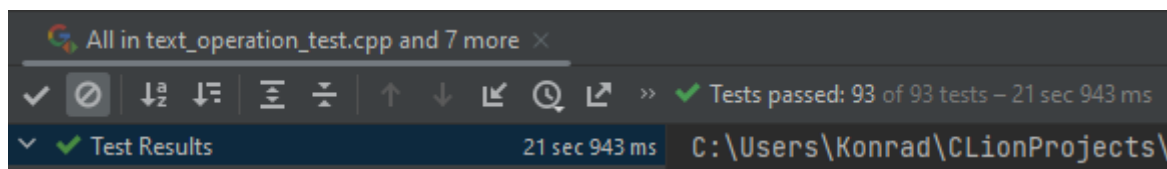
Rozwiązanie zostało napisane w języku C++ od podstaw bez użycia gotowych bibliotek implementujących tego typu rozwiązanie. Do zaimplementowania programu zostały użyte biblioteki i narzędzia.

- Cmake
- GoogleTests
- GCC

Testy

Do testowania oprogramowania wybraliśmy bibliotekę Google Test. Każdy moduł w celu potwierdzenia działania według założeń posiada testy jednostkowe. Forma testów jednostkowych z racji czasu jest ograniczona głównie do pozytywnych przypadków jednakże są też testy sprawdzające błędne użycia. Jednakże na tym polu przydałoby się dopisać brakujące testy. Ostatecznie testy pomogły sprawdzić i wyłapać sporo błędów.

Wynikowo zostały napisane 93 testy jednostkowe.



Kod rozwiązania opis ogólny

Zostały stworzone następujące moduły

Cnumpy – odpowiednik pythonowej biblioteki numpy. W naszej wersji obsługuje ona na razie macierze dwuwymiarowe. Brakuje jej wiele funkcjonalności które nie zostały napisane. Nie jest sprawdzana implementacja pod kątem wydajnościowym z powodu tego że nie przyjęliśmy założeń aby to robić. (Ograniczony czas). W przyszłości można się nad tym zastanowić. Obsługiwane funkcjonalności. (Łącznie kilkadziesiąt metod) to:

- Wczytywanie do Cnumpy plików CSV
- Budowanie dowolnego rozmiaru macierzy 2 wymiarowych
- Operacje matematyczne i statystyczne
- Ustawianie , pobieranie wartości elementów, kolumn
- Obsługa błędów

Decision tree – klasa odpowiedzialna za budowę drzewa decyzyjnego i wykonywanie predykcji.

CSV – Odczyt danych csv z pliku

Entropy – Obliczanie entropii informacji i zysku informacyjnego

Collection_Utility – Metody narzędziowe dla kolekcji

I pozostałe moduły wewnętrzne używane przez Cnumpy takie jak np. quant,math,text.

Cnumpy przykład użycia. Wczytanie danych z pliku csv i wyświetlenie ich na konsoli.

```
Cnumpy data = csv_reader.read_cnumpy_from_csv( path_to_file: "C:\\Users\\Konrad\\Documents\\repo\\randomforrest\\random
std::cout<<data;
```

	sepal.length Double 0	sepal.width Double 1	petal.length Integer 2	petal.width Double 3	variety String 4
0	6.1	2.8	4	1.3	Versicolor
1	5.2	4.1	1	0.1	Setosa
2	5.1	3.5	1	0.2	Setosa
3	5.8	2.8	5	2.4	Virginica
4	5.8	2.8	5	2.4	Virginica
5	6.5	3	5	2.2	Virginica
6	5.1	2.5	3	1.1	Versicolor
7	5.7	3.8	1	0.3	Setosa
8	6.7	3	5	2.3	Virginica
9	4.7	3.2	1	0.2	Setosa
10	5.8	2.7	5	1.9	Virginica
11	5.1	3.8	1	0.3	Setosa
12	5.5	2.5	4	1.3	Versicolor
13	5.8	2.8	5	2.4	Virginica

Ostatecznie Cnumpy pozwala na w miarę elastyczne modelowanie różnych modeli uczenia maszynowego z pomocą jego użycia zaimplementowaliśmy nasz model lasu losowego.

Rozwiązywany problem

Dla naszego testu rozwiązania spróbowaliśmy wykorzystać zbiór danych iris, kanoniczny zbiór danych uczenia maszynowego. Chcieliśmy do niego dodać coś od siebie więc dodaliśmy dwie dodatkowe grupy kwiatów do tego zbioru danych (Stokrotkę i Jaskra).

Opis zbioru danych

Zestaw pomiarów kwiatów irysa, udostępniony po raz pierwszy przez Ronalda Fishera w roku 1936. Jeden z najbardziej znanych zbiorów, a zarazem bardzo prosty i użyteczny. Celem jest wytrenowanie systemu, który na podstawie 4 podanych parametrów, poda właściwą klasę kwiatu (jedną z trzech dostępnych). My dodaliśmy tutaj po 50 rekordów dla stokrotki i jaskra.

Ilość próbek	Ilość atrybutów	Brakujące wartości	Problem
250	5	Brak	Klasyfikacja

Przykładowe wiersze danych

```
sepal length,sepal width,petal length,petal width,class  
5.1,3.5,1.4,0.2,Iris-setosa  
4.9,3.0,1.4,0.2,Iris-setosa  
4.7,3.2,1.3,0.2,Iris-setosa  
7.0,3.2,4.7,1.4,Iris-versicolor  
6.4,3.2,4.5,1.5,Iris-versicolor  
6.9,3.1,4.9,1.5,Iris-versicolor  
6.3,3.3,6.0,2.5,Iris-virginica  
5.8,2.7,5.1,1.9,Iris-virginica  
7.1,3.0,5.9,2.1,Iris-virginica  
6.3,2.9,5.6,1.8,Iris-virginicacs
```

Kilka przykładowych wierszy które dodaliśmy do oryginalnego zbioru iris.

```
0.5,0.1,1.2,0.7,"Ranunculus"  
0.6,0.2,1.0,0.7,"Ranunculus"  
0.5,0.3,0.7,0.5,"Ranunculus"  
0.5,0.2,0.6,0.1,"Bellis perennis"  
0.5,0.1,0.4,0.1,"Bellis perennis"
```

Wygląd kwiatów będących elementami zbioru
IRIS SETOSA



IRIS VERSICOLOR



IRIS VIRGINICA



Bellis perennis



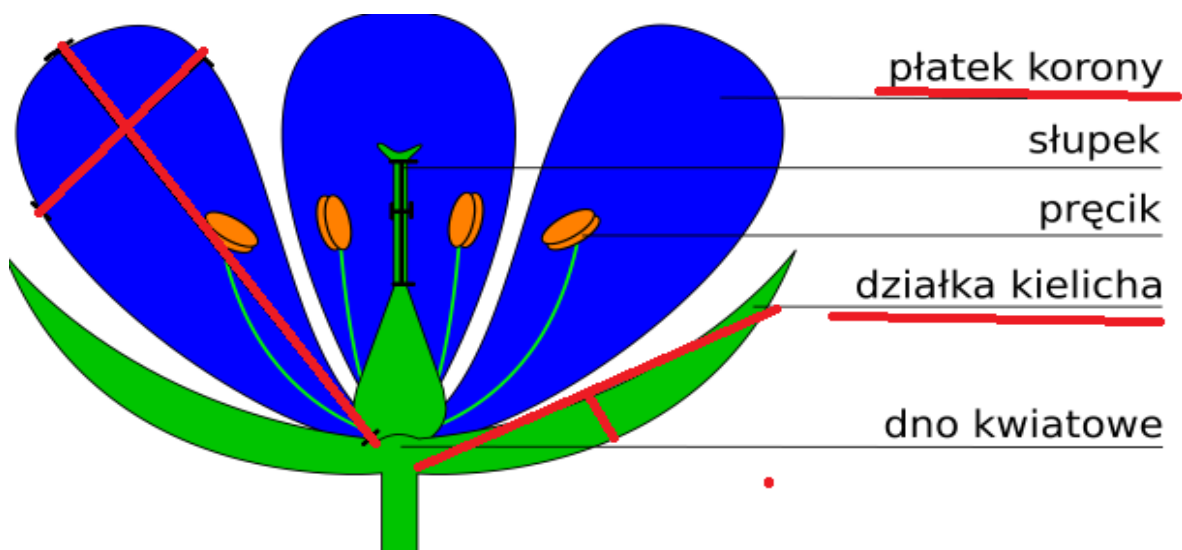
Ranunculus



Procedura pozyskania nowych danych

Zebraliśmy 100 kwiatów. 50 dla stokrotki i 50 dla jaskra. Dla każdego kwiatu zostały zmierzone. (Oznaczone na grafice poniżej)

sepal length - długość płatka korony
 sepal width - szerokość płatka korony
 petal length - długość działki kielicha
 petal width - szerokość działki kielicha



Przykład realizacji drzewa decyzyjnego

W celu budowy drzewa usunęliśmy z danych z których budujemy drzewo 15 wierszy tak aby zachować je w celu testowania skuteczności przewidywań modelu. Celem modelu jest podanie gatunku kwiatu na podstawie pomiarów opisanych powyżej.

Dane testowe

Dane testowe są 15 wierszami zabranymi z oryginalnego zbioru danych w celu sprawdzenia przewidywań modelu na nowych danych. Gdybyśmy tych rekordów nie usunęli ze zbioru to byśmy pokazywali programowi wiersze na podstawie których się uczył przez co miałby on łatwiej. Dodatkowo cechą drzew decyzyjnych, przynajmniej tej implementacji ID3 jest to że jeśli dane są zrównoważone (a zazwyczaj są) zna od odpowiedzi na każdy rekord uczący.

```
5.0,3.6,1.4,.2,"Setosa"  
5.4,3.7,1.5,.2,"Setosa"  
5.8,4.0,1.2,.2,"Setosa"  
6.0,2.2,4.0,1.0,"Versicolor"  
6.3,2.5,4.9,1.5,"Versicolor"  
6.8,2.8,4.8,1.4,"Versicolor"  
6.5,3.0,5.5,1.8,"Virginica"  
7.7,3.8,6.7,2.2,"Virginica"  
7.7,2.6,6.9,2.3,"Virginica"  
0.5,0.1,0.6,0.6,"Ranunculus"  
0.5,0.2,0.6,0.6,"Ranunculus"  
0.7,0.2,0.9,0.8,"Ranunculus"  
0.5,0.1,0.5,0.1,"Bellis perennis"  
0.4,0.1,0.5,0.1,"Bellis perennis"  
0.5,0.1,0.7,0.1,"Bellis perennis"
```

Kod testujący rozwiązanie

W funkcji main wywołujemy jedną funkcję w której jest zaszyta nasza logika

```
int main() {  
  
    check_one_tree_prediction();  
  
}
```

W podanej funkcji

- 1- Wczytujemy dane z pliku który jest zapisany w zmiennej globalnej IRIS_SUBSET ma on wycięte 15 wierszy które są naszym zestawem testowym.
- 2- Wyświetlamy całe wczytane dane na konsolę
- 3- Tworzymy na podstawie danych drzewo decyzyjne którego kolumna numer 4 (indeksowanie od zera) jest naszym atrybutem oznaczającym gatunek kwiatka. (predykcyjnym)
- 4- W tej funkcji tworzymy ręcznie Cnumpy z zawartością 15 testowych wierszy podanych wyżej
- 5- Tutaj obliczamy procent poprawnych predykcji dla zbioru testowego
- 6- Wyświetlamy procent poprawnych predykcji
- 7- Tworzymy Cnumpy z wartością w kolumnach opisującą nasze prognozy i oczekiwania
- 8- Wyświetlamy Cnumpy

```

void check_one_tree_prediction(){
    set_global_strategy_for_cnumpy();

    decision_tree tree;
    csv csv_reader;

    Cnumpy data = csv_reader.read_cnumpy_from_csv( path_to_file: path_to_file::IRIS_SUBSET, delimiter: ",");//1
    std::cout<<data; //2
    tree_node root = tree.construct_general_tree(data, predict_column_index: 4);//3

    Cnumpy test_data = create_test_datasets(); //4
    double percent_correct = count_effectiveness( datasets: test_data,root,tree);//5
    std::cout<<"Model Efficiency = "<<percent_correct<<std::endl;//6
    Cnumpy result_label = compare_result_to_expected_result( datasets: test_data,root,tree);//7
    std::cout<<result_label<<std::endl;//8
}

```

Kod pozostałych funkcji i całego rozwiązania dostępny w serwisie GitHub.

Dane wyjściowe

Całe dane dla algorytmu. Formatowanie zepsute przez Word

	sepal.length Double 0	sepal.width Double 1	petal.length Double 2	petal.width Double 4	variety String

0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.4	3.9	1.7	0.4	Setosa
5	4.6	3.4	1.4	0.3	Setosa
6	5	3.4	1.5	0.2	Setosa
7	4.4	2.9	1.4	0.2	Setosa
8	4.9	3.1	1.5	0.1	Setosa
9	4.8	3.4	1.6	0.2	Setosa
10	4.8	3	1.4	0.1	Setosa
11	4.3	3	1.1	0.1	Setosa
12	5.7	4.4	1.5	0.4	Setosa
13	5.4	3.9	1.3	0.4	Setosa
14	5.1	3.5	1.4	0.3	Setosa
15	5.7	3.8	1.7	0.3	Setosa
16	5.1	3.8	1.5	0.3	Setosa
17	5.4	3.4	1.7	0.2	Setosa
18	5.1	3.7	1.5	0.4	Setosa
19	4.6	3.6	1	0.2	Setosa
20	5.1	3.3	1.7	0.5	Setosa
21	4.8	3.4	1.9	0.2	Setosa
22	5	3	1.6	0.2	Setosa
23	5	3.4	1.6	0.4	Setosa

24	5.2	3.5	1.5	0.2	Setosa
25	5.2	3.4	1.4	0.2	Setosa
26	4.7	3.2	1.6	0.2	Setosa
27	4.8	3.1	1.6	0.2	Setosa
28	5.4	3.4	1.5	0.4	Setosa
29	5.2	4.1	1.5	0.1	Setosa
30	5.5	4.2	1.4	0.2	Setosa
31	4.9	3.1	1.5	0.2	Setosa
32	5	3.2	1.2	0.2	Setosa
33	5.5	3.5	1.3	0.2	Setosa
34	4.9	3.6	1.4	0.1	Setosa
35	4.4	3	1.3	0.2	Setosa
36	5.1	3.4	1.5	0.2	Setosa
37	5	3.5	1.3	0.3	Setosa
38	4.5	2.3	1.3	0.3	Setosa
39	4.4	3.2	1.3	0.2	Setosa
40	5	3.5	1.6	0.6	Setosa
41	5.1	3.8	1.9	0.4	Setosa
42	4.8	3	1.4	0.3	Setosa
43	5.1	3.8	1.6	0.2	Setosa
44	4.6	3.2	1.4	0.2	Setosa
45	5.3	3.7	1.5	0.2	Setosa
46	5	3.3	1.4	0.2	Setosa
47	7	3.2	4.7	1.4	Versicolor
48	6.4	3.2	4.5	1.5	Versicolor
49	6.9	3.1	4.9	1.5	Versicolor
50	5.5	2.3	4	1.3	Versicolor
51	6.5	2.8	4.6	1.5	Versicolor
52	5.7	2.8	4.5	1.3	Versicolor
53	6.3	3.3	4.7	1.6	Versicolor
54	4.9	2.4	3.3	1	Versicolor
55	6.6	2.9	4.6	1.3	Versicolor
56	5.2	2.7	3.9	1.4	Versicolor
57	5	2	3.5	1	Versicolor
58	5.9	3	4.2	1.5	Versicolor
59	6.1	2.9	4.7	1.4	Versicolor
60	5.6	2.9	3.6	1.3	Versicolor
61	6.7	3.1	4.4	1.4	Versicolor
62	5.6	3	4.5	1.5	Versicolor
63	5.8	2.7	4.1	1	Versicolor
64	6.2	2.2	4.5	1.5	Versicolor
65	5.6	2.5	3.9	1.1	Versicolor
66	5.9	3.2	4.8	1.8	Versicolor
67	6.1	2.8	4	1.3	Versicolor
68	6.1	2.8	4.7	1.2	Versicolor
69	6.4	2.9	4.3	1.3	Versicolor
70	6.6	3	4.4	1.4	Versicolor
71	6.7	3	5	1.7	Versicolor
72	6	2.9	4.5	1.5	Versicolor
73	5.7	2.6	3.5	1	Versicolor

74	5.5	2.4	3.8	1.1	Versicolor
75	5.5	2.4	3.7	1	Versicolor
76	5.8	2.7	3.9	1.2	Versicolor
77	6	2.7	5.1	1.6	Versicolor
78	5.4	3	4.5	1.5	Versicolor
79	6	3.4	4.5	1.6	Versicolor
80	6.7	3.1	4.7	1.5	Versicolor
81	6.3	2.3	4.4	1.3	Versicolor
82	5.6	3	4.1	1.3	Versicolor
83	5.5	2.5	4	1.3	Versicolor
84	5.5	2.6	4.4	1.2	Versicolor
85	6.1	3	4.6	1.4	Versicolor
86	5.8	2.6	4	1.2	Versicolor
87	5	2.3	3.3	1	Versicolor
88	5.6	2.7	4.2	1.3	Versicolor
89	5.7	3	4.2	1.2	Versicolor
90	5.7	2.9	4.2	1.3	Versicolor
91	6.2	2.9	4.3	1.3	Versicolor
92	5.1	2.5	3	1.1	Versicolor
93	5.7	2.8	4.1	1.3	Versicolor
94	6.3	3.3	6	2.5	Virginica
95	5.8	2.7	5.1	1.9	Virginica
96	7.1	3	5.9	2.1	Virginica
97	6.3	2.9	5.6	1.8	Virginica
98	6.5	3	5.8	2.2	Virginica
99	7.6	3	6.6	2.1	Virginica
100	4.9	2.5	4.5	1.7	Virginica
101	7.3	2.9	6.3	1.8	Virginica
102	6.7	2.5	5.8	1.8	Virginica
103	7.2	3.6	6.1	2.5	Virginica
104	6.5	3.2	5.1	2	Virginica
105	6.4	2.7	5.3	1.9	Virginica
106	6.8	3	5.5	2.1	Virginica
107	5.7	2.5	5	2	Virginica
108	5.8	2.8	5.1	2.4	Virginica
109	6.4	3.2	5.3	2.3	Virginica
110	6	2.2	5	1.5	Virginica
111	6.9	3.2	5.7	2.3	Virginica
112	5.6	2.8	4.9	2	Virginica
113	7.7	2.8	6.7	2	Virginica
114	6.3	2.7	4.9	1.8	Virginica
115	6.7	3.3	5.7	2.1	Virginica
116	7.2	3.2	6	1.8	Virginica
117	6.2	2.8	4.8	1.8	Virginica
118	6.1	3	4.9	1.8	Virginica
119	6.4	2.8	5.6	2.1	Virginica
120	7.2	3	5.8	1.6	Virginica
121	7.4	2.8	6.1	1.9	Virginica
122	7.9	3.8	6.4	2	Virginica
123	6.4	2.8	5.6	2.2	Virginica

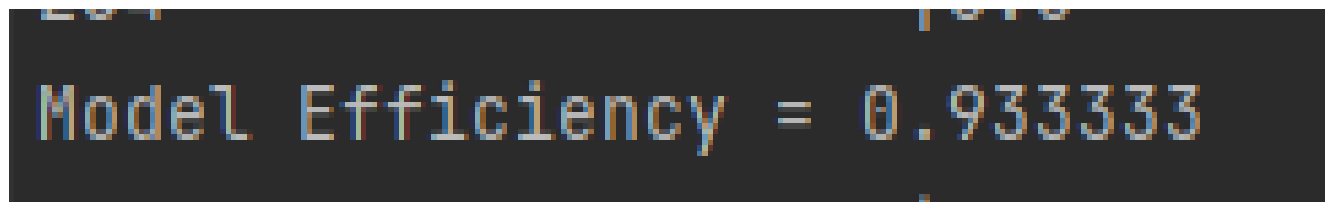
124	6.3	2.8	5.1	1.5	Virginica
125	6.1	2.6	5.6	1.4	Virginica
126	7.7	3	6.1	2.3	Virginica
127	6.3	3.4	5.6	2.4	Virginica
128	6.4	3.1	5.5	1.8	Virginica
129	6	3	4.8	1.8	Virginica
130	6.9	3.1	5.4	2.1	Virginica
131	6.7	3.1	5.6	2.4	Virginica
132	6.9	3.1	5.1	2.3	Virginica
133	5.8	2.7	5.1	1.9	Virginica
134	6.8	3.2	5.9	2.3	Virginica
135	6.7	3.3	5.7	2.5	Virginica
136	6.7	3	5.2	2.3	Virginica
137	6.3	2.5	5	1.9	Virginica
138	6.5	3	5.2	2	Virginica
139	6.2	3.4	5.4	2.3	Virginica
140	5.9	3	5.1	1.8	Virginica
141	0.6	0.4	0.8	0.6	Ranunculus
142	0.6	0.2	0.9	0.5	Ranunculus
143	0.6	0.1	1	0.8	Ranunculus
144	0.5	0.3	0.5	0.5	Ranunculus
145	0.7	0.3	0.8	0.7	Ranunculus
146	0.6	0.3	0.9	0.6	Ranunculus
147	0.5	0.2	1	0.7	Ranunculus
148	0.6	0.2	0.9	0.6	Ranunculus
149	0.6	0.3	0.06	0.5	Ranunculus
150	0.5	0.2	1	0.5	Ranunculus
151	0.7	0.2	0.9	0.7	Ranunculus
152	0.5	0.2	0.6	0.6	Ranunculus
153	0.5	0.2	0.6	0.5	Ranunculus
154	0.5	0.2	0.6	0.5	Ranunculus
155	0.6	0.2	0.9	0.6	Ranunculus
156	0.5	0.2	0.9	0.5	Ranunculus
157	0.6	0.2	0.5	0.5	Ranunculus
158	0.4	0.2	0.5	0.5	Ranunculus
159	0.5	0.2	0.6	0.5	Ranunculus
160	0.6	0.2	0.7	0.6	Ranunculus
161	0.5	0.1	0.6	0.5	Ranunculus
162	0.5	0.1	0.6	0.5	Ranunculus
163	0.6	0.3	0.6	0.4	Ranunculus
164	0.6	0.3	1	0.8	Ranunculus
165	0.5	0.2	0.8	0.8	Ranunculus
166	0.6	0.2	0.9	0.6	Ranunculus
167	0.6	0.2	0.8	0.6	Ranunculus
168	0.6	0.3	0.8	0.5	Ranunculus
169	0.6	0.4	0.8	0.6	Ranunculus
170	0.5	0.2	0.7	0.4	Ranunculus
171	0.5	0.2	0.7	0.5	Ranunculus
172	0.5	0.1	0.9	0.6	Ranunculus
173	0.5	0.2	0.5	0.4	Ranunculus

174	0.7	0.3	0.5	0.6	Ranunculus
175	0.5	0.2	0.6	0.5	Ranunculus
176	0.5	0.3	0.7	0.5	Ranunculus
177	0.5	0.2	1	0.6	Ranunculus
178	0.4	0.1	0.6	0.6	Ranunculus
179	0.5	0.2	1	0.7	Ranunculus
180	0.4	0.2	0.5	0.4	Ranunculus
181	0.6	0.2	0.7	0.5	Ranunculus
182	0.5	0.1	1	0.6	Ranunculus
183	0.6	0.2	1	0.6	Ranunculus
184	0.6	0.3	1.2	0.7	Ranunculus
185	0.5	0.1	1.2	0.7	Ranunculus
186	0.6	0.2	1	0.7	Ranunculus
187	0.5	0.3	0.7	0.5	Ranunculus
188	0.5	0.2	0.6	0.1	Bellis perennis
189	0.5	0.1	0.4	0.1	Bellis perennis
190	0.6	0.1	0.4	0.1	Bellis perennis
191	0.5	0.2	0.8	0.1	Bellis perennis
192	0.7	0.1	0.9	0.1	Bellis perennis
193	0.6	0.1	0.6	0.1	Bellis perennis
194	0.5	0.2	0.5	0.1	Bellis perennis
195	0.6	0.2	0.5	0.1	Bellis perennis
196	0.5	0.1	0.7	0.1	Bellis perennis
197	0.4	0.1	0.6	0.1	Bellis perennis
198	0.5	0.2	0.6	0.1	Bellis perennis
199	0.5	0.2	0.8	0.1	Bellis perennis
200	0.5	0.1	0.6	0.1	Bellis perennis
201	0.5	0.1	0.5	0.1	Bellis perennis
202	0.5	0.2	0.5	0.1	Bellis perennis
203	0.6	0.1	0.6	0.1	Bellis perennis
204	0.5	0.2	0.7	0.1	Bellis perennis
205	0.5	0.1	0.4	0.1	Bellis perennis
206	0.4	0.1	0.6	0.1	Bellis perennis
207	0.5	0.1	0.5	0.1	Bellis perennis
208	0.5	0.2	0.4	0.1	Bellis perennis
209	0.5	0.1	0.6	0.1	Bellis perennis
210	0.4	0.1	0.6	0.1	Bellis perennis
211	0.4	0.1	0.3	0.1	Bellis perennis
212	0.5	0.2	0.6	0.1	Bellis perennis
213	0.5	0.1	0.5	0.1	Bellis perennis
214	0.5	0.1	0.6	0.1	Bellis perennis
215	0.6	0.1	0.8	0.1	Bellis perennis
216	0.5	0.1	0.5	0.1	Bellis perennis
217	0.5	0.1	0.7	0.1	Bellis perennis
218	0.5	0.2	0.7	0.1	Bellis perennis
219	0.5	0.1	0.7	0.1	Bellis perennis
220	0.4	0.1	0.6	0.1	Bellis perennis
221	0.5	0.1	0.6	0.1	Bellis perennis
222	0.5	0.2	0.6	0.1	Bellis perennis
223	0.5	0.2	0.6	0.1	Bellis perennis

224	0.5	0.1	0.6	0.1	Bellis perennis
225	0.5	0.1	0.6	0.1	Bellis perennis
226	0.5	0.1	0.5	0.1	Bellis perennis
227	0.4	0.1	0.6	0.1	Bellis perennis
228	0.6	0.1	0.7	0.1	Bellis perennis
229	0.3	0.1	0.5	0.1	Bellis perennis
230	0.6	0.1	0.8	0.1	Bellis perennis
231	0.6	0.2	0.6	0.1	Bellis perennis
232	0.7	0.2	0.6	0.1	Bellis perennis
233	0.5	0.1	0.9	0.1	Bellis perennis
234	0.5	0.2	0.5	0.1	Bellis perennis

Skuteczność modelu

Widzimy że nasz model dokonał poprawnej predykcji dla 14 z 15 wierszy 😊



Model Efficiency = 0.933333

Macierz pomyłek

Tutaj widzimy że nasz model pomylił irys z gatunku Versicolor, ściślej mówiąc nie był w stanie wydedukować poprawnego wyniku. (W takiej sytuacji najczęściej losuje się wynik z dostępnych opcji lub stosuje jakąś bardziej wyrafinowaną metodę).

Dygresja

Jako ciekawostkę mogę podać że tak jak każdy algorytm uczenia maszynowego jest on mocno zależny od ilości danych uczących więcej/lepiej. Ostatecznie każdy nawet najprostszy (w skali racjonalności) model jest w stanie działać bardzo dobrze wystarczy dać mu jak najwięcej danych. Generalnie same wybory modeli, budowanie rozwiązań odbywa się w celu przeciwdziałania małej ilości danych. Gdybyśmy mieli wystarczającą ilość danych każdy model byłby dobry. Jest to jeden z czynników dla których uczenie maszynowe i sztuczna inteligencja mają więcej możliwości do pokazania się szerszej publiczności bo mamy coraz więcej danych i większą moc obliczeniową która pozwala przetwarzać te dane w coraz większych ilościach. Pozwalając sobie na trochę prywaty uważam za fascynujące obserwowanie jak te modele zmieniają świat. Dostęp w przyszłości do takiego urządzenia jak komputer kwantowy spowoduje praktycznie nieograniczoną moc obliczeniową co oznacza że w gruncie rzeczy modele (mimo wszystko jednak w pewnych elementach losowe) jakie teraz stosujemy będą mogły zostać problemem NP.-trudnym gdzie po prostu przeszukamy całe przestrzenie rozwiązań i zrobimy to z łatwością pomimo tego iż oryginalnie jest on NP.-trudny. Wydaje mi się iż jest to nieunikniona ścieżka rozwoju ludzkości, fascynująca ale

jednocześnie mogąca być wielkim zagrożeniem dla ludzkości ale gdybyśmy nie ryzykowali stalibyśmy w miejscu.

	Expected String 0	Predict String 1	STATUS String 2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Versicolor	MATCH
5	Versicolor	BRAK	MISMATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Czas

To raczej ciekawostka z racji tego że pomiar czasu w ten sposób nie powinien być traktowany z ufnością. (Obciążenie komputera, niedokładność mechanizmów itd). Podany czas mierzyliśmy za pomocą biblioteki chrono. Kod całego naszego rozwiązania Cnumpy itd. Jest jednowątkowy.

Komputer na którym był mierzony czas

Procesor Intel core i7 9700f

Pamięć RAM 16 GB

Dysk twardy Kingston KC3000

Próba numer	Czas
1	49101547 microseconds (49.10s)
2	48322202 microseconds (48.32s)
3	48186962 microseconds (48.19s)

Lasy losowe

Las losowy jest metodą zespołową uczenia maszynowego dla klasyfikacji, regresji i innych zadań, która polega na konstruowaniu wielu drzew decyzyjnych w czasie uczenia i generowaniu klasy, która jest dominantą klas (klasyfikacja) lub przewidywaną średnią (regresja) poszczególnych drzew. Losowe lasy decyzyjne poprawiają tendencję drzew decyzyjnych do nadmiernego dopasowywania się do zestawu treningowego.

My zrealizowaliśmy tylko model klasyfikacyjny.

Model zespołowy

Założymy że zadajesz trudne pytanie kilkuset osobą a następnie zbierasz odpowiedzi i łączysz je w jedną wynikową. Taka odpowiedź statystycznie będzie lepsza niż jakbyś zadał jednej losowej osobie to pytanie. W lasach losowych tworzymy n drzew decyzyjnych na podstawie wybranych wierszy danych uczących za pomocą mechanizmu losowania ze zwracaniem. Zbieramy predykcje która klasa występuje najczęściej ją wybieramy jako naszą predykcję.

Przykład naszej realizacji dla przykładu w którym używaliśmy 10 drzew decyzyjnych w lesie.

Kod rozwiązania

Tutaj również w funkcji main wywołujemy naszą funkcję. Tutaj pierwszym argumentem jest ilość drzew w lesie losowym, drugim (tutaj mocno za małym 😊) argumentem jest ilość linii dostępna dla każdego drzewa jako dane uczące.

```
int main() {  
    check_forrest_prediction( number_tree: 10, number_of_line: 15);  
}
```

Opis głównej funkcji

Całość tych operacji jest wykonywana w pętli poza początkowym stworzeniem podzbiorów uczących dla każdego drzewa. W pętli tworzymy model każdego drzewa i wykonujemy za jego pomocą predykcje i zbieramy wyniki.

1. Tutaj za pomocą klasy narzędziowej tworzymy zbiór danych dla każdego z naszych drzew
2. Tutaj tworzymy obiekt klasy naszego drzewa
3. Wczytujemy dane dla odpowiedniego drzewa
4. Konstruujemy drzewo na podstawie odczytanych danych
5. Tworzymy nasz zbiór testowy zawierający 15 wierszy
6. Porównujemy predykcje naszego drzewa z oczekiwanymi rezultatami

Dalej znajduje się jeszcze kod który wyświetla ładne podsumowanie predykcji (reszta kodu w serwisie GitHub)

```

void check_forrest_prediction(int number_tree,int number_of_line){
    set_global_strategy_for_cnumpy();

    create_data_set create_dataset;

    //1
    create_dataset.create_files( numberOfFiles: number_tree, read_file: path_to_file::IRIS_SUBSET, numberOfLine: number_of_line);
    std::vector<Cnumpy> predicts_all;

    for(int i=0;i<number_tree;++i){
        //2
        decision_tree tree;
        csv csv_reader;
        //3
        Cnumpy data = csv_reader.read_cnumpy_from_csv( path_to_file: "C:\\Users\\Konrad\\CLionProjects\\randomforest\\data\\iris_subset.csv");
        //4
        tree_node root = tree.construct_general_tree(data, predict_column_index: 4);
        //5
        Cnumpy test_data = create_test_datasets();
        //6
        Cnumpy result_label = compare_result_to_expected_result( datasets: test_data,root,tree); //to
        predicts_all.push_back(result_label); //to
    }
}

```

Testy rozwiązania

Spróbujmy wytrenować 10 drzew dając każdemu po 215 elementów jako dane uczące.

Wyniki

Wyniki dla poszczególnych drzew kształtują się następująco. Są to dane wyjściowe z programu formatowanie zepsute przez word w wynikowym programie wygląda to następująco. (Przykład dla drzewa numer 10). Na zielono zaznaczyłem poprawne predykcje drzew.

Drzewo numer10 wyniki			
	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	BRAK	MISMATCH
7	Virginica	BRAK	MISMATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 1 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Virginica	MISMATCH
5	Versicolor	Virginica	MISMATCH
6	Virginica	BRAK	MISMATCH
7	Virginica	BRAK	MISMATCH
8	Virginica	BRAK	MISMATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 2 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Versicolor	MATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 3 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Versicolor	MATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	BRAK	MISMATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 4 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 5 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	BRAK	MISMATCH
10	Ranunculus	BRAK	MISMATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 6 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Versicolor	MATCH
5	Versicolor	BRAK	MISMATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 7 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	Virginica	MATCH
7	Virginica	BRAK	MISMATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 8 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 9 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	Versicolor	MATCH
5	Versicolor	BRAK	MISMATCH
6	Virginica	Virginica	MATCH
7	Virginica	Virginica	MATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Drzewo numer 10 wyniki

	Expected	Predict	STATUS
	String	String	String
	0	1	2
0	Setosa	Setosa	MATCH
1	Setosa	Setosa	MATCH
2	Setosa	Setosa	MATCH
3	Versicolor	Versicolor	MATCH
4	Versicolor	BRAK	MISMATCH
5	Versicolor	Versicolor	MATCH
6	Virginica	BRAK	MISMATCH
7	Virginica	BRAK	MISMATCH
8	Virginica	Virginica	MATCH
9	Ranunculus	Ranunculus	MATCH
10	Ranunculus	Ranunculus	MATCH
11	Ranunculus	Ranunculus	MATCH
12	Bellis perennis	Bellis perennis	MATCH
13	Bellis perennis	Bellis perennis	MATCH
14	Bellis perennis	Bellis perennis	MATCH

Finalne predykcje

WYNIKI CAŁOŚCI	Expected String 0	EMPTY Integer 1	Setosa Integer 2	Versicolor Integer 3	Virginica Integer 4	Ranunculus Integer 5	Bellis perennis Integer 6	Predicted String 7
0	Setosa	0	10	0	0	0	0	Setosa
1	Setosa	0	10	0	0	0	0	Setosa
2	Setosa	0	10	0	0	0	0	Setosa
3	Versicolor	0	0	10	0	0	0	Versicolor
4	Versicolor	5	0	4	1	0	0	Setosa
5	Versicolor	2	0	7	1	0	0	Versicolor
6	Virginica	3	0	0	7	0	0	Virginica
7	Virginica	3	0	0	7	0	0	Virginica
8	Virginica	1	0	0	9	0	0	Virginica
9	Ranunculus	1	0	0	0	9	0	Ranunculus
10	Ranunculus	1	0	0	0	9	0	Ranunculus
11	Ranunculus	0	0	0	0	10	0	Ranunculus
12	Bellis perennis	0	0	0	0	0	10	Bellis perennis
13	Bellis perennis	0	0	0	0	0	10	Bellis perennis
14	Bellis perennis	0	0	0	0	0	10	Bellis perennis

Predicted	Expected String 0	EMPTY Integer 1	Setosa Integer 2	Versicolor Integer 3	Virginica Integer 4	Ranunculus Integer 5	Bellis perennis Integer 6	Bellis perennis String 7
0	Setosa	0	10	0	0	0	0	Setosa
1	Setosa	0	10	0	0	0	0	Setosa
2	Setosa	0	10	0	0	0	0	Setosa
3	Versicolor	0	0	10	0	0	0	Versicolor
4	Versicolor	5	0	4	1	0	0	Setosa
5	Versicolor	2	0	7	1	0	0	Versicolor
6	Virginica	3	0	0	7	0	0	Virginica
7	Virginica	3	0	0	7	0	0	Virginica
8	Virginica	1	0	0	9	0	0	Virginica
9	Ranunculus	1	0	0	0	9	0	Ranunculus
10	Ranunculus	1	0	0	0	9	0	Ranunculus
11	Ranunculus	0	0	0	0	10	0	Ranunculus
12	Bellis perennis	0	0	0	0	0	10	Bellis perennis
13	Bellis perennis	0	0	0	0	0	10	Bellis perennis
14	Bellis perennis	0	0	0	0	0	10	Bellis perennis

Na zielono widzimy wiersze w których każde drzewo zagłosowało poprawnie. Na zielono i pomarańczowo widzimy poprawnie sklasyfikowane wiersze. Na czerwono jest jedyny źle sklasyfikowany wiersz. Dla jednego drzewa które analizowaliśmy wcześniej też ten wiersz był źle sklasyfikowany.

Dla tego wiersza drzewa głosowały następująco

Puste -5

Setosa-0

Versicolor -4

Virginica -1

Ranunculus -0

Bellis perennis -0

Drzewo zagłosowało na Setosę z powodu wybranego przez nas systemu traktowania wyboru EMPTY jako największej wartości. Gdy tak jest losujemy wynik z możliwych klas. Biorąc pod uwagę następną klasę po EMPTY w takim przypadku mielibyśmy rację bo poprawna klasa to Versicolor. Zmiana strategii wydaje się optymalnym podejściem. Jeszcze lepszym podejściem jest zebranie większej ilości danych. Z racji braku czasu nie przetestowaliśmy

regularyzacji i innego sposobu głosowania ale wydają się to ciekawe aspekty do sprawdzenia. (Nie mierzyliśmy też większej ilości kwiatów 😊) Same drzewa decyzyjne są modelem wyjątkowym w uczeniu maszynowym pod pewnymi względami, mają one jasną dla ludzi ścieżkę predykcji i praktycznie nie wymagają regularyzacji. Nie sposób powiedzieć że pomimo dodania przez nas własnych danych ten zbiór danych jest raczej łatwym i nasz algorytm dla 250 wierszy danych wejściowych uzyskuje bardzo dobre wyniki poprawność wynoszącą 93 %. Jest to taki sam wynik jak dla jednego drzewa ale wynika to z prostoty zbioru danych. Pomimo tej prostoty przy zmianie kryterium głosowania mielibyśmy 100% poprawności. Ogólnie lasy losowe uzyskują lepsze wyniki od pojedynczych drzew. Postaram się to przedstawić niżej analizując mniejszą ilość wierszy z danych przypadającą na poszczególne drzewo.

```
Skuteczność modelu to 0.933333
```

Czas

Wynikowy czas predykcji naszego lasu losowego jako średnia z trzech przebiegów to 389020960 mikrosekund czyli 389.02s. Około 6 i pół minuty. Jest to w zasadzie stałe z ilością drzew * czas konstrukcji jednego drzewa. Ponieważ nasz program jest jedno wątkowy przez co wykonuje tworzenie drzew po kolei.

Złożoność obliczeniowa

Szukanie optymalnego drzewa jest problemem NP.-trudnym. Często stosuje się tu heurystyki złożoność predykcji wynosi średnio $O(n)$. Każdy poziom drzewa wymaga sprawdzenia wartości atrybutu. W przypadku drzew binarnych z biblioteki scikit-learn jest to $O(\log_2(n))$. Uczenie drzewa ma złożoność w przypadku wzorcowej implementacji $O(nxm \log_2(m))$.

W ramach projektu przetestowaliśmy tylko algorytm konstrukcji drzewa ID3 ale istnieje ich wiele jak choćby CART zastosowany w scikit-learn. ID3 charakteryzuje się następującymi zaletami i wadami.

Wady i zalety algorytmu ID3

Zalety

- prostota
- Jeżeli w danych uczących nie ma hałasu, czyli że nie ma rekordów które dla tych samych wartości atrybutów mają przypisaną różną kategorię, to ID3 daje poprawny wynik dla wszystkich rekordów ze zbioru treningowego. (Pokazane jest to w pierwszym rozpisany przykładzie jak to działa)

Wady

- nie radzi sobie z ciągłymi dziedzinami atrybutów (zakłada, że wartości atrybutów są dyskretne). Można tutaj skwantować dane i próbować je przedyktować. To też można by sprawdzić nawet w naszej teraźniejszej implementacji ponieważ w Cnumpy napisaliśmy metodę umożliwiającą kwantowanie danych.
- zakłada sztywno, że wszystkie rekordy w danych są wypełnione tzn. nie działa, jeżeli choć jeden rekord zawiera niepełne dane.

- duży rozmiar drzewa
- brak odporności na zjawisko zwane overfitting. Polega to na tym, że algorytm nie radzi sobie z danymi zaburzającymi ogólną ich informację. Może to w rezultacie prowadzić do wysokiego współczynnika błędów na danych testowych.

Eksperymenty

Teraz przeprowadzimy eksperymenty z ilością drzew i wierszy danych uczących dla każdego drzewa.

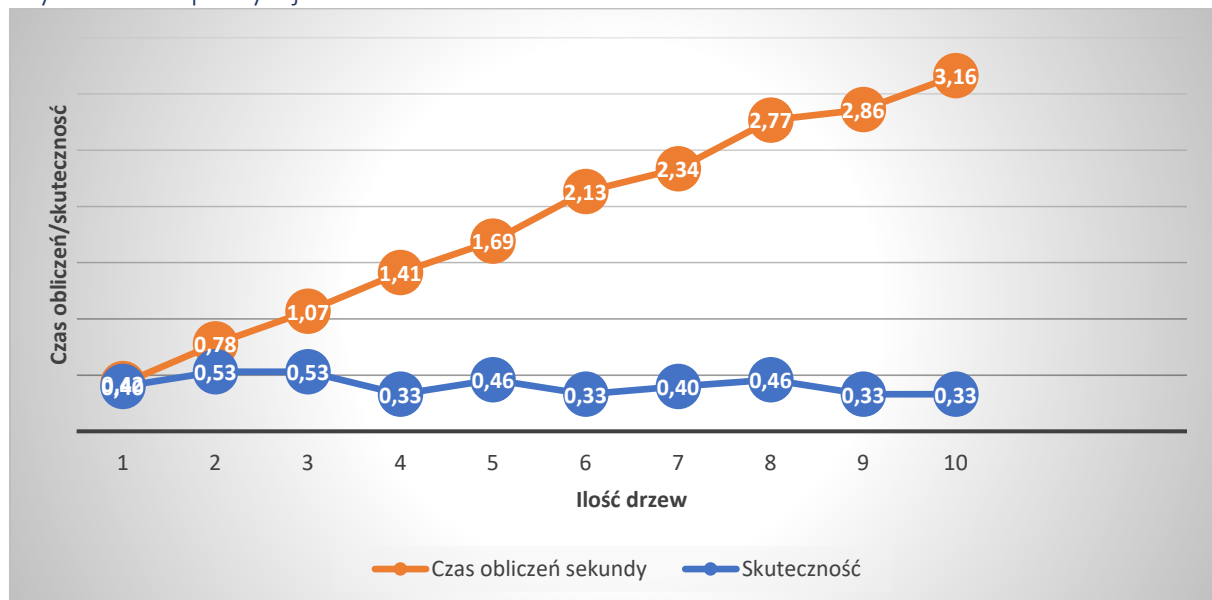
Uczenie lasu z 15 wierszy danych uczących na każde drzewo

W pierwszym eksperymencie wybierzemy tylko 15 wierszy jako dane treningowe dla każdego drzewa. Jest to zdecydowanie za mało ale jest ciekawym eksperymentem pod kątem ewentualnej poprawy predykcji w skutek zwiększenia ilości drzew w lesie.

Czasy budowy i predykcji

Każde drzewo z 15 wierszy			
Ilość drzew	Czas obliczeń sekundy		Skuteczność
1		0,42	0,40
2		0,78	0,53
3		1,07	0,53
4		1,41	0,33
5		1,69	0,46
6		2,13	0,33
7		2,34	0,40
8		2,77	0,46
9		2,86	0,33
10		3,16	0,33

Wykres czasu predykcji i skuteczności modelu.



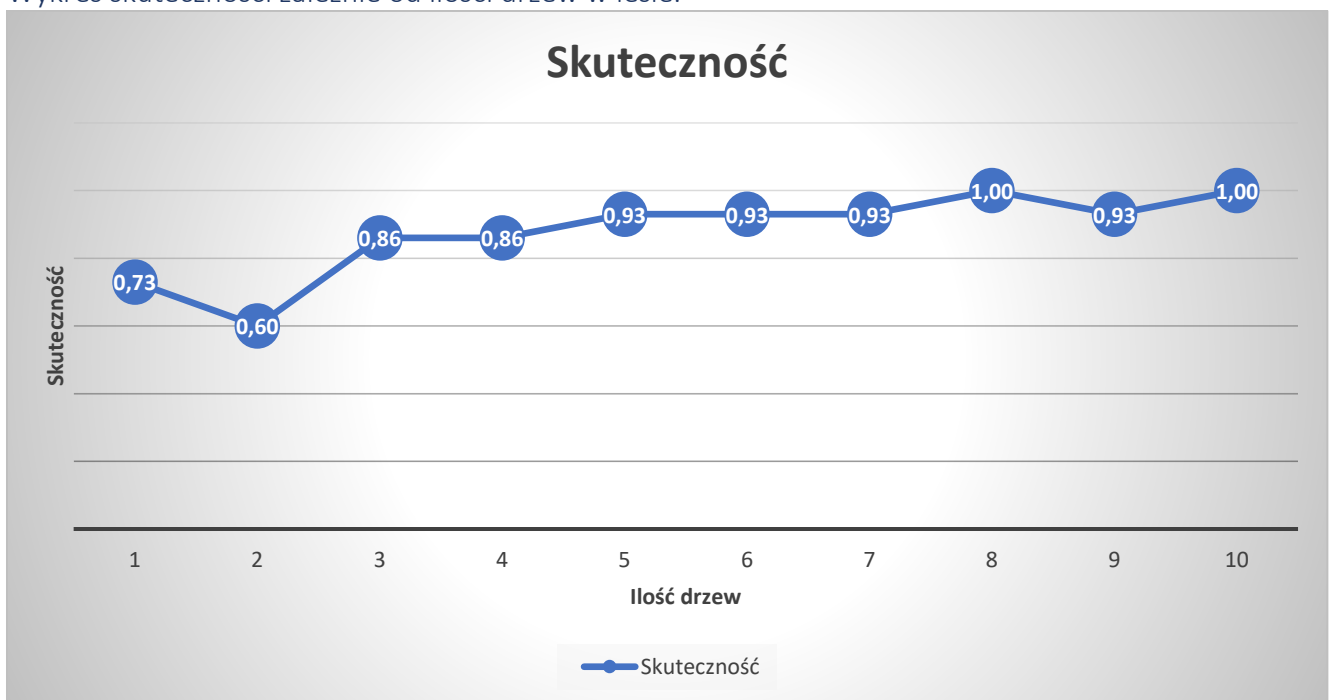
Jak widać czas obliczeń w stosunku do ilości drzew ma tendencję liniową. Natomiast Skuteczność naszego modelu cały czas oscyluje w granicy od 30 do 50 procent. Ilość drzew wydaje się nie mieć na to wpływu. Powodem jest niedotrenowanie modelu (Dajcie mi dane 😊). Każde nasze drzewo posiada tylko 15 wierszy danych uczących przez co nie jest w stanie zapewnić lepszej skuteczności.

Uczenie lasu z 80 wierszy na każde drzewo

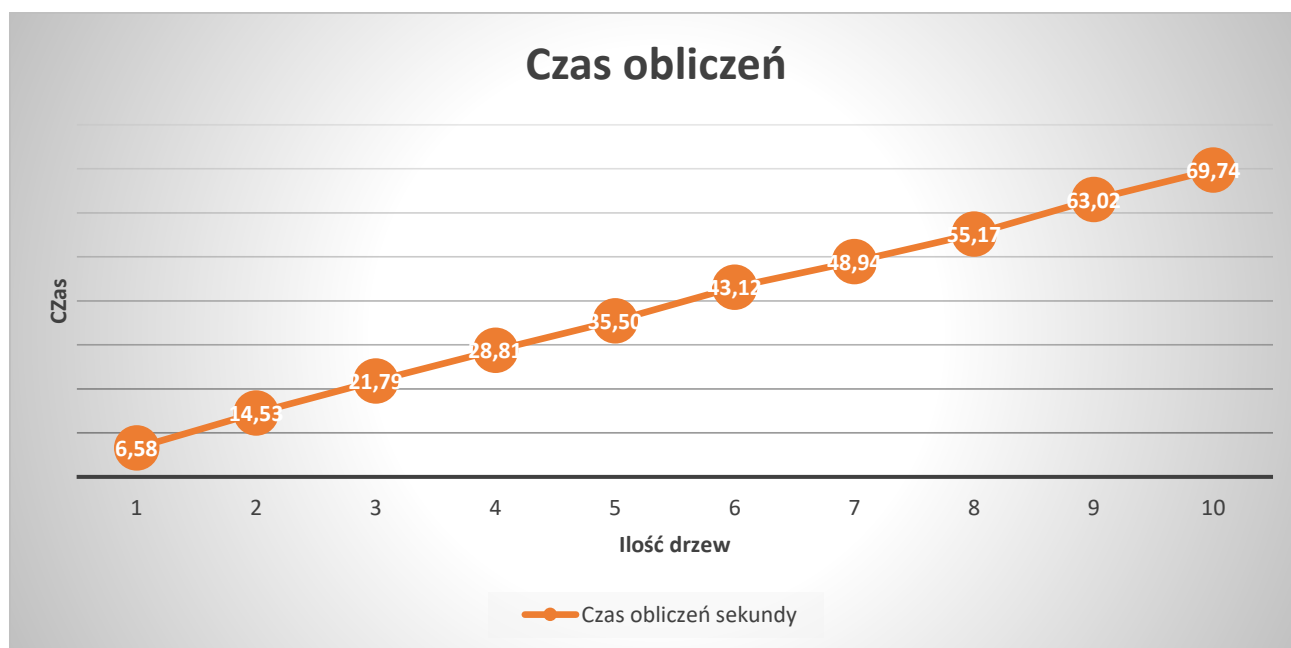
Teraz dajmy szansę naszym lasom. Dla każdego drzewa damy po 80 przykładów uczących. Oto wyniki dla drzew od 1 do 10. Widać tutaj że dalej wzrost czasu obliczeń jest liniowy analogicznie do tego co się spodziewaliśmy. Natomiast widać tu coś co dla 15 przykładów na drzewo było niemożliwe do osiągnięcia czyli poprawę skuteczności w zależności od liczby drzew w lesie. Otrzymaliśmy nawet magiczną 100 procentową poprawność. Co jest związane z prostotą zbioru danych ale zawsze to coś.

Kaźde drzewo z 80 wierszy		
Ilość drzew	Czas obliczeń sekundy	Skuteczność
1	6,58	0,73
2	14,53	0,60
3	21,79	0,86
4	28,81	0,86
5	35,50	0,93
6	43,12	0,93
7	48,94	0,93
8	55,17	1,00
9	63,02	0,93
10	69,74	1,00

Wykres skuteczności zależnie od ilości drzew w lesie.



Wykres czasu obliczeń w stosunku do ilości drzew



Podsumowanie

Dzięki zastosowaniu podejścia gdzie całą funkcjonalność lasów losowych wraz z niezbędnymi narzędziami zaimplementowaliśmy od zera udało nam się uzyskać ich dość dobre zrozumienie. Pyzatem dzięki utworzeniu Cnumpy mamy prostą drogę do potencjalnego rozszerzania naszego kodu o inne modele uczenia maszynowego. Same lasy losowe zaimplementowane przez nas mogą modelować dowolny problem. Tutaj akurat zajęliśmy się rozpoznawaniem kwiatów.

Cześć 2 Środowisko PVM

W ramach projektu z racji tego żeby być zgodnymi z wymaganiami zaimplementowaliśmy wykorzystanie naszego modelu lasu losowego w środowisku PVM. Same środowisko jest nie wspierane od dłuższego czasu więc tutaj skupiliśmy się raczej na przetestowaniu naszego rozwiązania aniżeli na budowie kompletnego uniwersalnego kodu. Stąd brak ogólnych mechanizmów pozwalających rozwiązywać dowolny problem za pomocą lasów losowych. Ograniczyliśmy się tutaj raczej do naszego problemu rozpoznawania kwiatów. Brakuje mechanizmu przesyłania Cnumpy na inne maszyny i „ładnej” prezentacji wyników za pomocą Cnumpy. Uznaliśmy za bezcelowe implementowanie podanej funkcjonalności. Po drodze natrafiliśmy na wiele problemów gdzie zależnie od obciążenia PVM potrafił przestawać działać bez powodu(nie chciał dodawać węzłów, po prostu nie zwracał wyników i wiele więcej). Wydaje nam się że są to problemy związane ze środowiskiem PVM a nie ściśle naszą implementacją. Ale nie inwestowaliśmy w to więcej czasu bo to miało się z celem.

Opis środowiska PVM

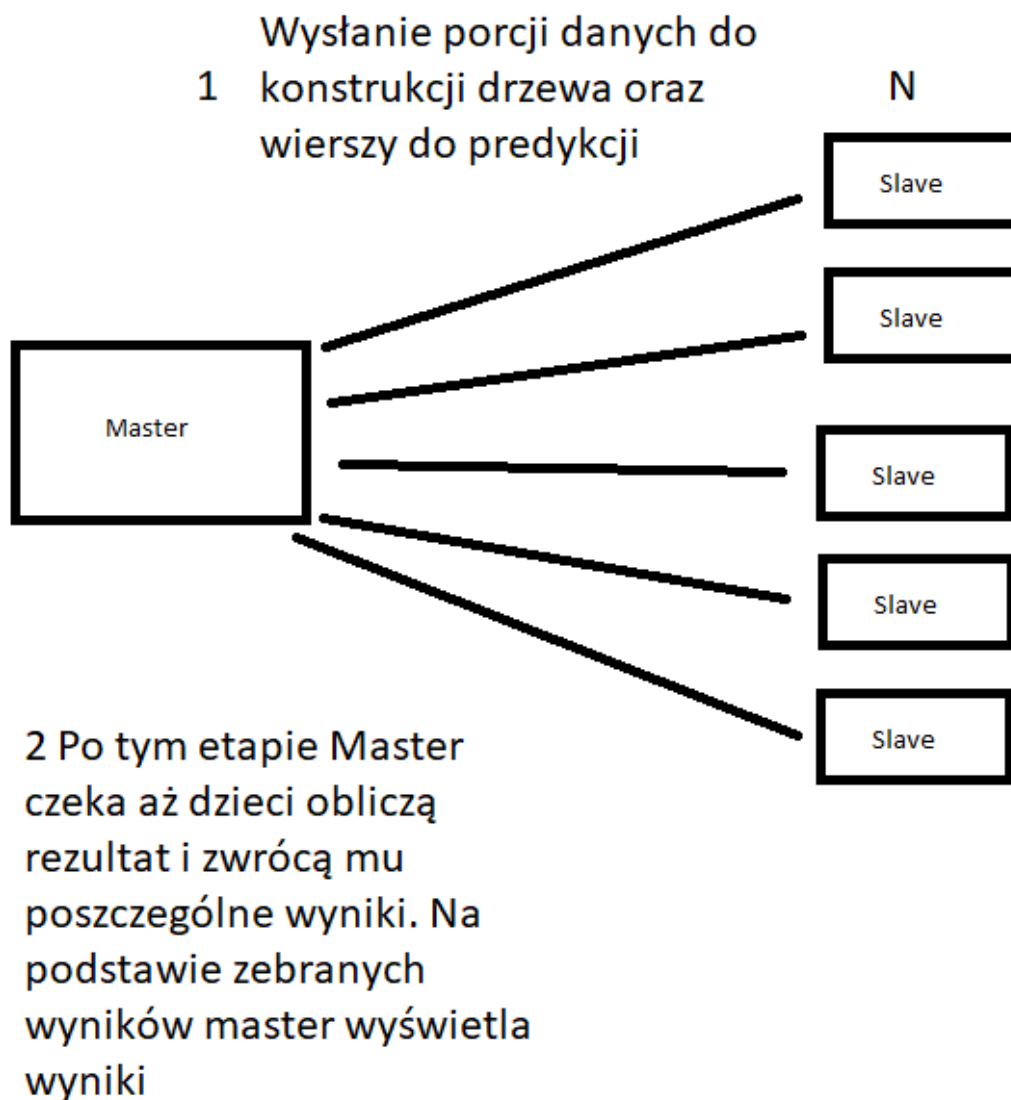
Pvm jest zestawem narzędzi do tworzenia oprogramowania dla sieci równolegle połączonych komputerów. Został zaprojektowany i stworzony by umożliwić łączenie komputerów o różnych konfiguracjach sprzętowych w jeden równolegle działający komputer.

PVM jest narzędziem służącym głównie do pośrednictwa w wymianie informacji pomiędzy procesami uruchomionymi na oddzielnych maszynach oraz do zarządzania nimi za pośrednictwem konsoli pvm. Z punktu widzenia obecnej definicji maszyny wirtualnej, pvm w sumie nie zasługuje na swoją nazwę, powinien się bardziej nazywać „zarządca procesów zrównoleglonych”, jednakże historia pvm sięga dawnych czasów, kiedy to nazewnictwo było nieco inaczej rozumiane.

W dużym uproszczeniu PVM pozwala na „połączenie” większej ilości komputerów w jeden. Odbywa się to na zasadzie dołączania kolejnych hostów (komputerów), na których został zainstalowany i skonfigurowany pvm. Podłączanie hosta jest w uproszczeniu procedurą połączenia przez rsh, uruchomieniu demona pvm i dostarczenia mu informacji o tym, kto jest jego „rodzicem” oraz o parametrach istniejącej sieci.

Architektura rozwiązania

Tak jest opisane na rysunku poniżej tworzymy n dzieci i wysyłamy im losowy (ze zwracaniem) podzbiór z x elementów danych. Na każdym dziecku tworzymy drzewa decyzyjne w odpowiedniej ustalonej w formie hiper parametru ilości ogólnie. Na każdym dziecku powstaje n drzew zależnie od PVM. Każde dziecko oblicza wynik po czym zwraca odpowiedź do rodzica który czeka na wyniki z wszystkich dzieci i gdy tylko je otrzyma wyświetla rezultat.



[Kod rozwiązania](#)

Cały kod w serwisie GitHub tutaj tylko omówimy go trochę

Master

Z argumentów linii poleceń odczytujemy hosty których chcemy użyć w programie.

```
int info[HOST_NUMBER];  
static char *hosts[] = { "", "", "", "", "", "", "", "", "", "", "", "", "", "", "" };  
for (int i = 0; i < HOST_NUMBER; i++)  
{  
    hosts[i] = argv[i + 2];  
}
```

Potem tworzymy dzieci w których będą tworzone poszczególne drzewa i przeprowadzane predykcje.

```
if (0 >=
    (taskNo =
        pvm_spawn("slave", NULL, PvmTaskDefault, "", TASK_NUMBER, tIds))) {
    printf("[%x] nie moge stworzyc dzieci\n", myId);
    pvm_perror("pvm_spawn");
    pvm_exit();
    return 1;
}
```

Potem wczytujemy zbiór uczący i 15 przykładów testowych na których testujemy skuteczność naszego modelu.

```
std::vector< std::string > readData = create_data.read_from_file(fileName);
int numberOfLine = 215;
std::vector< std::string > dataRead = create_data.read_from_file("dataToRead.csv");
double tab[15][4] =
    {{0,0,0,0}, {0,0,0,0},{0,0,0,0},
     {0,0,0,0} ,{0,0,0,0},{0,0,0,0},
     {0,0,0,0} ,{0,0,0,0},{0,0,0,0},
     {0,0,0,0} ,{0,0,0,0},{0,0,0,0},
     {0,0,0,0} ,{0,0,0,0}, {0,0,0,0}
    };

for (int i = 0; i < 15; i++)
{
    std::vector<std::string> temp = split(dataRead[i] ,",");
    for (int j = 0; j < 4; j++)
    {
        tab[i][j] = std::stod(temp[j]);
    }
}
```

Następnie tworzymy zbiory z danych uczących dla poszczególnych drzew i przesyłamy je do odpowiednich dzieci. Przesyłamy również wiersze do predykcji.

```

printf("[%x] stworzyłem pomyslnie %i procesy\n", myId, taskNo);
for (int i = 0; i < taskNo; i++) {
    std::string dataToSend;
    std::vector < std::string > split_data_to_file = create_data.split_data(readData, numberOfLine);

    dataToSend += firstLine + "\n";
    for (int j = 0; j < numberOfLine; j++)
    {
        dataToSend += split_data_to_file[j] + "\n";
    }

    pvm_initsend(PvmDataDefault);
    pvm_packf("%s", dataToSend.data());
    pvm_send(tIds[i], 2);

    for (int j = 0; j < 15; j++) {
        pvm_initsend(PvmDataDefault);
        pvm_pkdouble(tab[j], 4, 1);
        pvm_send(tIds[i], 3 + j);
    }
}

```

Na koniec odczytujemy odpowiedzi dzieci dla danych testowych

```

std::string results;
for (int i = 0; i < taskNo; i++) {
    pvm_initsend(PvmDataDefault);
    pvm_recv(tIds[i], 50);
    pvm_unpackf("%s", bufor[i]);
    std::string result(bufor[i]);
    std::cout << i << " " << result << std::endl;
    results += result;
}

```

Reszta kodu odpowiada za wyświetlenie wyników.

Sleave

W kodzie dzieci odczytujemy dane uczące. Zapisujemy do pliku. Potem otwieramy podany plik za pomocą utworzonych przez nas mechanizmów, konstruujemy drzewo decyzyjne. Odczytujemy również dane testowe.

```

int bajtow, msgTag, tId, bufId;
double tab[15][4];
create_data_set create_data;
printf("[%x] Dziecko: moj rodzic %x\n", myId, parentId);
pvm_initsend(PvmDataDefault);
pvm_recv(parentId, 2 );
pvm_unpackf("%s", bufor);
std::string dataToSend(bufor);

create_data.write_to_file(dataToSend, std::to_string(myId) + "data.csv");
set_global_strategy_for_cnumpy();
decision_tree tree;
csv csv_reader;
Cnumpy data = csv_reader.read_cnumpy_from_csv(std::to_string(myId) + "data.csv", ",");
tree_node root = tree.construct_general_tree(data, 4);
for (int j = 0; j < 15; j++) {
    pvm_initsend(PvmDataDefault);
    pvm_recv(parentId, 3 + j);
    pvm_upkdouble(tab[j], 4, 1);
}
std::string cnmpayResults;

```

Potem za pomocą utworzonego drzewa wykonujemy klasyfikacje danych testowych i wysyłamy odpowiedź do rodzica.

```

std::string cnmpayResults;
for (int i = 0; i < 15; i++) {
    std::vector<Type> columns_in_iris_datasets{Type::double_type, Type::double_type, Type::double_type, Type::double_type, Type::string_type};

    Cnumpy row(5, 1, columns_in_iris_datasets);
    row.set(0, 0, tab[i][0]);
    row.set(1, 0, tab[i][1]);
    row.set(2, 0, tab[i][2]);
    row.set(3, 0, tab[i][3]);
    row.set(4, 0, "test");
    Cnumpy cnmpayResult = tree.predict(root, row);
    cnmpayResults += cnmpayResult.get_xy_string(0,0) + "\n";
}

std::cout << "DZIECKO" << myId << " " << cnmpayResults << std::endl;
pvm_initsend(PvmDataDefault);
pvm_packf("%s", cnmpayResults.data());
pvm_send(parentId, 50);

```

Przykład uruchomienia kodu

Zrzut z uruchomienia programu na 2 maszynach wirtualnych.

```
Liczba wezlow: 2
Liczba architektur: 1
Wezel marekm ma architekture LINUX64 i predkosc 1000
Wezel konradg ma architekture LINUX64 i predkosc 1000
[40002] startuje
[t40003] BEGIN
[t40004] BEGIN
[t40005] BEGIN
[t40006] BEGIN
[t40007] BEGIN
[t40008] BEGIN
[t40009] BEGIN
[t80001] BEGIN
[t80002] BEGIN
[t80003] BEGIN
[t80004] BEGIN
[t80005] BEGIN
[t80006] BEGIN
[t80007] BEGIN
[t80008] BEGIN
TEST
[40002] stworzylem pomyslnie 15 procesy
[t40006] [40006] startuje
[t40006] [40006] Dziecko: moj rodzic 40002
[t40006] DZIECKO262150 Setosa
[t40006] Setosa
[t40006] Versicolor
[t40006] Versicolor
[t40006] Versicolor
[t40006] Versicolor
[t40006] BRAK!!!
[t40006] BRAK!!!
[t40006] Virginica
[t40006] Versicolor
[t40006] Versicolor
[t40006] Versicolor
[t40006] Versicolor
[t40006] Versicolor
```

Predykcje zwracane przez program dla poszczególnych wierszy testowych

```
sepal.length: 5.4 sepal.width: 3.7 petal.length: 1.5 petal.width: 0.2  
: 1  
Setosa: 15  
  
sepal.length: 5.8 sepal.width: 4 petal.length: 1.2 petal.width: 0.2  
Ranunculus: 1  
Setosa: 14  
  
sepal.length: 6 sepal.width: 2.2 petal.length: 4 petal.width: 1  
Versicolor: 15  
  
sepal.length: 6.3 sepal.width: 2.5 petal.length: 4.9 petal.width: 1.5  
BRAK: 5  
Versicolor: 10  
  
sepal.length: 6.8 sepal.width: 2.8 petal.length: 4.8 petal.width: 1.4  
BRAK: 9  
Versicolor: 6  
  
sepal.length: 6.5 sepal.width: 3 petal.length: 5.5 petal.width: 1.8  
BRAK: 5  
Virginica: 10  
  
sepal.length: 7.7 sepal.width: 3.8 petal.length: 6.7 petal.width: 2.2  
BRAK: 4  
Virginica: 11  
  
sepal.length: 7.7 sepal.width: 2.6 petal.length: 6.9 petal.width: 2.3  
BRAK: 1  
Virginica: 14  
  
sepal.length: 8.5 sepal.width: 8.1 petal.length: 8.6 petal.width: 8.6  
BRAK: 1  
Ranunculus: 14  
  
sepal.length: 8.5 sepal.width: 8.2 petal.length: 8.6 petal.width: 8.6  
BRAK: 1  
Ranunculus: 14  
  
sepal.length: 8.7 sepal.width: 8.2 petal.length: 8.9 petal.width: 8.8  
Ranunculus: 15  
  
sepal.length: 8.5 sepal.width: 8.1 petal.length: 8.5 petal.width: 8.1  
Bellis perennis: 15  
  
sepal.length: 8.4 sepal.width: 8.1 petal.length: 8.5 petal.width: 8.1  
Bellis perennis: 15
```

Szybkość działania

Maszyny wirtualne

Pierwszy test działania programu przeprowadziliśmy na maszynach wirtualnych z pomocą oprogramowania VirtualBox. Środowiskiem na którym były stawiane maszyny wirtualne jest ten sam komputer którego specyfikacje podawaliśmy wyżej. Każda maszyna posiadała po 1 rdzeniu i 1 GB pamięci RAM. Podany kod różni się od tego który wcześniej uruchamialiśmy na jednym komputerze bez użycia środowiska PVM dlatego czasy wykonania mogą od siebie odbiegać znacząco. Dla przykładu tutaj testujemy budowę 15 drzew w lesie i kod odpowiedzialny za prezentację rezultatu jest inny z powodów opisanych wyżej. Wyniki uruchomień kodu prezentują się następująco (Kilkakrotne uruchomienie daje podobne wyniki)

1 Maszyna

```
sepal.length: 0.5 sepal.width: 0.2 petal.length: 0.6 petal.width: 0.6  
BRAK: 1  
Ranunculus: 14  
  
sepal.length: 0.7 sepal.width: 0.2 petal.length: 0.9 petal.width: 0.8  
Ranunculus: 15  
  
sepal.length: 0.5 sepal.width: 0.1 petal.length: 0.5 petal.width: 0.1  
Bellis perennis: 15  
  
sepal.length: 0.4 sepal.width: 0.1 petal.length: 0.5 petal.width: 0.1  
Bellis perennis: 15  
  
sepal.length: 0.5 sepal.width: 0.1 petal.length: 0.7 petal.width: 0.1  
Bellis perennis: 15  
  
sepal.length: 0.5 sepal.width: 0.2 petal.length: 0.5 petal.width: 0.1  
Bellis perennis: 15  
  
Liczba hostow 1  
Time: 637.125 seconds.  
[t4000e] EOF
```

2 Maszyny

```
Liczba wezlow: 2  
Liczba architektur: 1  
Wezel konradg ma architekture LINUX64 i predkosc 1000  
Wezel konradg2 ma architekture LINUX64 i predkosc 1000  
[t4002d] startuje
```

```
Liczba hostow 2  
Time: 435.631 seconds.  
[t40069] EOF
```

Tabela obrazująca czasy wykonania kodu na jednej i dwóch maszynach.

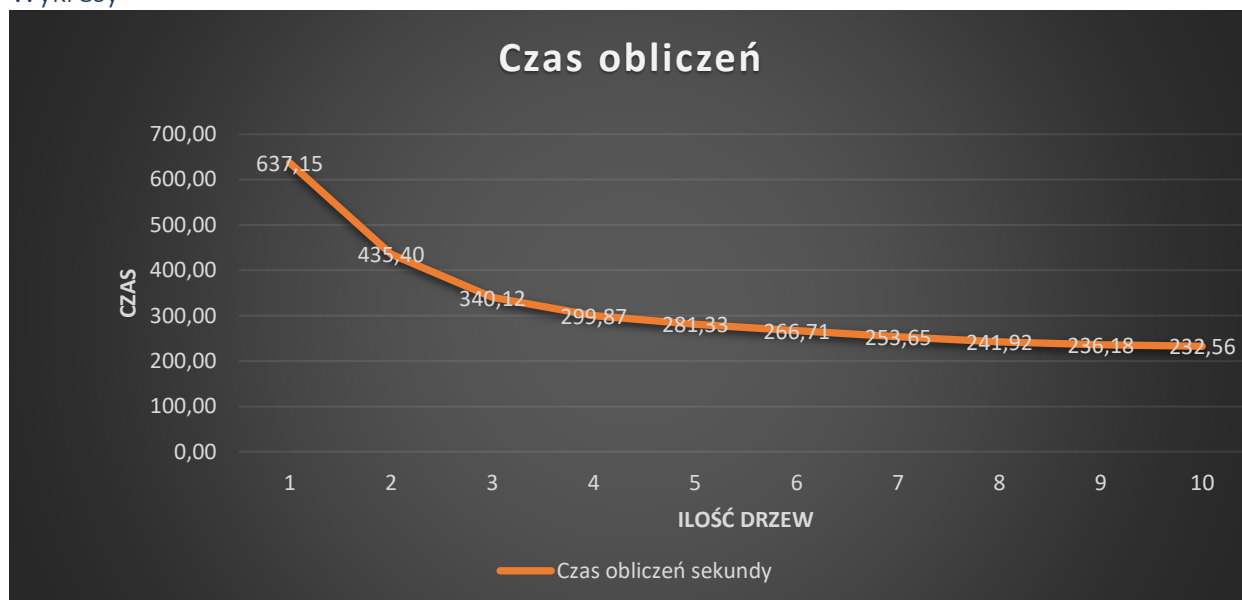
Ilość maszyn wirtualnych	Czas wykonania (s)
1	637.15
2	435.40

Jak się spodziewaliśmy program dla 2 maszyn działa szybciej z powodu tego że każda maszyna oblicza wyniki dla 5 drzew równolegle z tą pierwszą a nie jak ma to miejsce w standardowej implementacji gdzie wszystkie 10 drzew tworzonych jest na jednej maszynie.

Testy na fizycznych komputerach

Każde drzewo z 215 wierszy (maszyny wirtualne) 15 drzew		
Ilość maszyn	Czas obliczeń sekundy	
1		637,15
2		435,40
3		340,12
4		299,87
5		281,33
6		266,71
7		253,65
8		241,92
9		236,18
10		232,56

Wykresy



Początkowy wzrost szybkości wykonania jest dość spory potem wraz z ilością urządzeń na których kod jest uruchamiany równoległe ten zysk spada. Co jest zrozumiałe ponieważ koszty przesyłu danych niwelują nam zysk otrzymywany z kolejnego urządzenia.

Wnioski

Udało nam się uruchomić nasze rozwiązanie na środowisku PVM. Udało nam się zaobserwować oczywisty trend zmian czasów wykonania programu w stosunku do ilości maszyn na których kod jest równoległe uruchamiany. Jednak samo działanie środowiska PVM pozostawia wiele do życzenia, jest to oprogramowanie praktycznie niezdatne do użytku w profesjonalnych rozwiązaniach z powodu częstych błędów nie związanych z kodem na nim uruchamianym.

Kolejne kroki

W ramach rozwiązania nie udało nam się z powodu ograniczeń czasowych przy realizacji projektu przetestować regularyzacji różnych form obliczeń zysku informacyjnego, różnych algorytmów konstrukcji drzew decyzyjnych. Przedstawienia krok po kroku przykładów jak to działa w naszym rozwiązaniu oraz wykonania wielu eksperymentów na danych i drzewach które przyszły nam do głowy. Zwyczajnie zabrakło nam czasu na ich wykonanie i opracowanie w ramach sprawozdania ale planujemy z własnej ciekawości je wykonać.

Bibliografia

https://pl.wikipedia.org/wiki/Parallel_Virtual_Machine

https://en.wikipedia.org/wiki/Parallel_Virtual_Machine

https://pl.wikipedia.org/wiki/Las_losowy