UMA - Projekt Końcowy

Ewa Roszczyk, Paweł Gryka

styczeń 2022

Spis treści

1 Temat projektu

Przygotować własny algorytm konstruujący drzewa klasyfikacyjne i zastosować go do przygotowania metody tworzącej ranking atrybutów. Ranking ma być tworzony przez wielokrotną konstrukcję drzewa na losowych podzbiorach zbioru uczącego oraz zliczenie częstości występowania poszczególnych atrybutów w testach.

2 Streszczenie projektu wstępnego

2.1 Wybrany algorytm

Szukając odpowiedniego algorytmu do zadania znaleźliśmy następujące:

- CART-Classification and Regression Trees
- ID3-Iterative Dichotomiser 3
- C4.5
- C5.0
- CHAID-Chi-squared Automatic Interaction Detection
- MARS

Zgodnie z dokumentacją wstępną zdecydowaliśmy się na algorytm C5.0. Decyzja taka została podjęta ze względu na to, iż jest on jednym z najbardziej znanych i cenionych algorytmów do implementacji drzew decyzyjnych. Algorytm jest udoskonaloną wersją swojego poprzednika - C4.5, który dla odmiany powstał na podstawie algorytmu ID3. Dodatkowo algorytm C5.0 jest szybszy, wydajniejszy pamięciowo oraz tworzy prostsze drzewa niż jego poprzednik - C4.5.

2.2 Wybrane zbiory

W ramach projektu wybranymi przez nas zbiorami były poniższe zbiory:

- Car evaluation - prostszy zbiór, posiada dyskretne dane oraz nie występują w nim braki danych.
- Adult - trudniejszy zbiór, posiada dane ciągłe oraz występują w nim braki danych.

Zgodnie z dokumentacją wstępną chcieliśmy, aby nasz algorytm mógł być wykorzystywany dla różnych zbiorów danych, co się udało.

2.3 Założenia implementacyjne

Poniżej znajdują się założenia implementacyjne z dokumentacji wstępnej:

- Implementacja w Pythonie
- Możliwość obsługi różnych zbiorów danych, nie tylko zbioru wybranego do testowania
- Konwersja wartości atrybutów ciągłych na dyskretne za pomocą podziału na zbiory
- Jeżeli w liściu dwie klasy mają taką samą szansę wystąpienia to wybierana jest losowa
- Opcja wyłączenia z zbioru S podzbioru S_i, w którym istnieją brakujące wartości atrybutów. Gdy podzbiór S_i jest włączany do trenowania, to do brakujących atrybutów wpisywane są najbardziej popularne wartości atrybutów z $S \backslash S_i$.
- Ranking atrybutów tworzony na postawie następującej metryki: Tworzymy licznik "ważności" każdego z atrybutów i inicjalizujemy każdy zerem. W momencie algorytmu gdy atrybut jest wybierany przez *attribute selection method*, czyli przez minimalną entropię, dodajemy do licznika moc aktualnie rozważanego podzbioru. Opisane kroki wykonamy dla drzew zbudowanych na różnych podzbiorach S. W ten sposób liczniki atrybutów posortowane malejąco stworzą ranking mówiący o tym ile danych dzieli dany atrybut.

2.4 Plan eksperymentów

Planujemy wykonać parę eksperymentów:

- Obliczanie metryk TPR, TNR, PPV, ACC, F1 score, oraz narysowanie ROC space dla każdego otrzymanego drzewa.
- Wykonanie wyżej wymienionych metod dla lasu stworzonego z n najlepszych drzew (decyzja o wybranej klasie przez głosowanie)
- Porównanie wyników i metryk dla zbiorów z dyskretnymi danymi a takich zawierających dane ciągłe.
- Porównanie wyników i metryk dla zbiorów zawierających brakujące wartości atrybutów a takimi, które są pełne (nie zawierają braków)

3 Realizacja zadana

3.1 Podstawowe informacje

Implementacje zrealizowaliśmy w języku python. Większość z założeń projektowych została spełniona, ze względu na natłok pracy, niestety nie zrealizowaliśmy tworzenia lasów losowych oraz rysowania krzywych ROC. Resztę założeń udało się spełnić, drzewa generowane są na podstawie dowolnych danych podzielonych na dwa pliki *data* i *names*. Generujemy rankingi atrybutów na dwa różne sposoby oraz sprawdzamy wydajność naszej implementacji - zarówno czasowe jak i metryki takie jak: TPR, TNR, PPV, ACC oraz F1 score dla każdej przewidywanej klasy.

3.2 Pełen opis funkcjonalny

Napisany przez nas program posiada kilka głównych funkcji:

- Wczytywanie dowolnych danych z plików csv (więcej w **sekcji o implementacji**)
- Uczenie modeli drzew decyzyjnych na wczytanych danych
- Przewidywanie klas próbek na podstawie wcześniej wygenereowanego modelu
- Przekształcanie drzew na plik json, który można pokazać w formie graficznej
- Obsługiwanie zarówno atrybutów dyskretnych jak i ciągłych
- Obliczanie podstawowych parametrów wydajnościowych wygenerowanych modeli
- Tworzenia rankingu atrybutów na podstawie wygenerowanego moedelu na dwa sposoby

Poniżej przedstawiamy przykładowe wyniki działania naszej implementacji generacji drzewa oraz predykcji na nim odpowiednio dla zbioru Car i Adult:

Wyniki dla zbioru Car z podziałem danych uczących/testujących w stosunku (4:3)

```
Correct = 687 | Incorrect = 54 | Overall_accuracy = 0.9271255060728745
    ========= unacc =========
        TP = 498 | FN = 24 | FP = 11 | TN = 189
        TPR = 0.9540229885057471
        TNR = 0.945
        PPV = 0.9783889980353635
        ACC = 0.9515235457063712
        F1 = 0.9660523763336566
    ========= acc =========
        TP = 147 | FN = 23 | FP = 23 | TN = 540
        TPR = 0.8647058823529412
        TNR = 0.9591474245115453
        PPV = 0.8647058823529412
        ACC = 0.937244201909959
        F1 = 0.8647058823529412
    ========= vgood =========
        TP = 24 | FN = 2 | FP = 9 | TN = 663
        TPR = 0.9230769230769231
        TNR = 0.9866071428571429
```

```
        PPV = 0.7272727272727273
        ACC = 0.9842406876790831
        F1 = 0.8135593220338984
    ========= good =========
        TP = 18 | FN = 5 | FP = 11 | TN = 669
        TPR = 0.782608695652174
        TNR = 0.9838235294117647
        PPV = 0.6206896551724138
        ACC = 0.9772403982930299
        F1 = 0.6923076923076923
 Error = 0
============= FINISHED =============
```

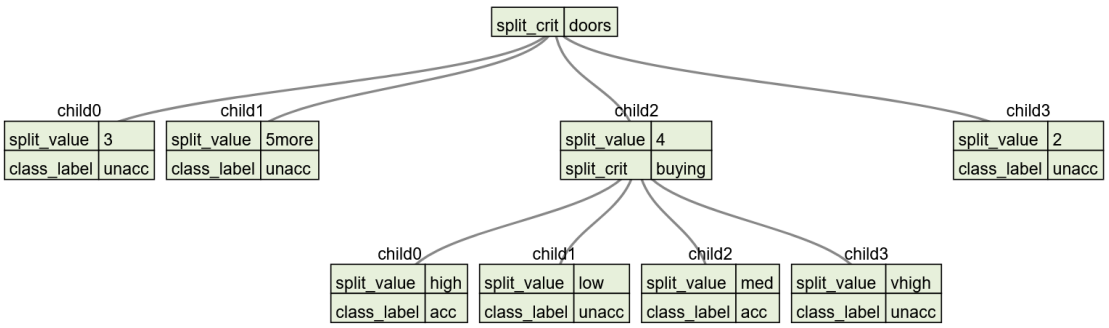
Wyniki dla zbioru Adult z podziałem danych uczących/testujących w stosunku (4:1)

```
Correct = 5291 | Incorrect = 1221 | Overall_accuracy = 0.8125
    =========  >50K =========
        TP = 926 | FN = 705 | FP = 516 | TN = 4365
        TPR = 0.5677498467198038
        TNR = 0.8942839582052858
        PPV = 0.6421636615811374
        ACC = 0.8125
        F1 = 0.6026684022128214
    =========  <=50K =========
        TP = 4365 | FN = 516 | FP = 705 | TN = 926
        TPR = 0.8942839582052858
        TNR = 0.5677498467198038
        PPV = 0.8609467455621301
        ACC = 0.8125
        F1 = 0.8772987639433223
 Error = 0
============= FINISHED =============
```

Jak można zaobserwować program automatycznie wyświetla obliczone metryki. O konkretnych wartościach tych metryk opowiadamy później.

Oprócz pokazanych powyżej wyników wyświetlana jest również reprezentacja drzewa w formacie json oraz link do **tej strony**, na której można to drzewo obejrzeć (po przeklejeniu json do odpowiedniego pola). Poniżej przedstawiamy przykładową zawartość json oraz wygląd drzewa dla bardzo małych danych trenujących:

```
{"split_crit": "doors", "child0": {"split_value": "3", "class_label": "unacc"},
↪  "child1": {"split_value": "5more", "class_labe l": "unacc"}, "child2":
↪  {"split_value": "4", "split_crit": "buying", "child0": {"split_value":
↪  "high", "class_label": "acc"}, "child1": {"split_value": "low",
↪  "class_label": "unacc"}, "child2": {"split_value": "med", "class_label":
↪  "acc"}, "child3": {"split_value": "vhigh", "class_label": "unacc"}},
↪  "child3": {"split_value": "2", "class_label": "unacc"}}
```



Rysunek 1: Przykładowe małe drzewo

Nie przedstawiamy tutaj wyniku dla dużych danych trenujących ponieważ po wklejeniu do dokumentu PDF jest on bardzo nieczytelny. Oferujemy za to możliwość zobaczenia reprezentacji graficznej takiego, pełnego drzewa. W tym celu należy skopiować cały tekst z **tego pliku** do pola "JSON" na **tej stronie**.

3.3 Implementacja rozwiązania

Pełną implementację oraz cały projekt znaleźć można na githubie pod **tym linkiem**.

3.3.1 Użyte biblioteki i narzędzia

W naszej implementacji użyliśmy następujących bibliotek:

- **pandas** - biblioteka do wygodnych operacji na zbiorach danych. Zaznaczyć należy, że biblioteki tej należy używać ostrożnie w projektach zawierających operacje na dużych zbiorach danych, potrafi ona bardzo spowolnić działanie algorytmów. Dzieje się tak, gdy nadużywamy funkcji iteracji po data-frame'ach, jest to bardzo nieefektywna i wolna operacja. Na szczęście biblioteka ma zaimplementowane wiele mechanizmów pozwalających na pominięcie iteracji po zbiorze, ale niestety nie zawsze jest to możliwe.
- **math** - biblioteka do obliczania logarytmów potrzebnych do obliczenia entropii.
- **matplotlib** - biblioteka do tworzenia wykresów.
- **numpy** - biblioteka służąca do obsługi dużych, wielowymiarowych tabel i macierzy.
- **statistics**
- **timeit**
- **json**
- **collections**

Użyte narzędzia:

- GitHub - repozytorium do przechowywania kodu (tutaj znajduje się nasz projekt),
- Visual Studio Code i PyCharm - edytory kody,
- Strona vanya - konwerter pliku JSON na diagram drzewa.

3.3.2 Najważniejsze sekcje kodu

Sercem naszego programu jest oczywiście sama implementacja algorytmu C5.0. Przedstawiamy ją poniżej:

```
 def generate_tree(self, S, attribute_dict=None):
        N = Node(parent_conn=None)
        if all_samples_of_same_class(S):
            class_name = S.iloc[0]["class"]
            return Node(parent_conn=None, class_label=class_name)
        if all_attributes_the_same_or_empty(S):
            return Node(parent_conn=None, class_label=find_most_common_class(S))
        n = self.attribute_selection_method(S)
        if attribute_dict is not None:
            attribute_dict[S.columns[n]] += S.shape[0]

        if S.columns[n] not in self.continuous_attributes:
            for value in unique(S.iloc[:, n]):
                N.split_crit = S.columns[n]
                # subset with an equal a_n value to "value"
                column_name = S.columns[n]
                S_n = S[S[column_name] == value]
                if S_n.empty:
                    attach_a_leaf(S, N, value)
                else:
```

```
                    self.attach_a_child(S_n.drop(S_n.columns[n], axis=1), N,
                    ↪  value, attribute_dict=attribute_dict)

        else:
            split_value = median(S.iloc[:, n])
            column_name = S.columns[n]
            N.split_crit = column_name

            # attach a child for bigger values
            S_n = S[S[column_name] > split_value]
            if S_n.empty:
                attach_a_leaf(S, N, f">{split_value}")
            else:
                self.attach_a_child(S_n.drop(S_n.columns[n], axis=1), N,
                ↪  f">{split_value}",
                                    attribute_dict=attribute_dict)

            # attach a child for smaller values
            S_n = S[S[column_name] <= split_value]
            if S_n.empty:
                attach_a_leaf(S, N, split_value)
            else:
                self.attach_a_child(S_n.drop(S_n.columns[n], axis=1), N,
                ↪  f"<={split_value}",
                                    attribute_dict=attribute_dict)

        return N
```

Implementacja jest bardzo podobna do pseudokodu prezentowanego w dokumentacji wstępnej. Główna różnica polega na dodaniu możliwości obsługi danych ciągłych. Przy tworzeniu drzewa podawana jest lista atrybutów o wartościach ciągłych (w szczególności może ona być pusta), która następnie jest wykorzystywana do decyzji jak podzielić dane względem danego atrybutu. Atrybuty o wartościach ciągłych dzielone są binarnie na te o wartości większych od mediany i resztę. Przedstawiony wyżej fragment kodu pochodzi z klasy **Tree**.

Uważamy, że nazwy funkcji wraz z komentarzami całkiem dobrze przedstawiają działanie kodu i, że wklejanie tutaj całego kodu jedynie zmniejszałby czytelność sprawozdania. Dlatego poniżej opisujemy pliki i klasy używane w projekcie a cały kod znajduje się na wspomnianym wcześniej **githubie**:

- **Tree** - serce projektu, zawiera metody generujące drzewo oraz "interfejs" do predykcji za jego pomocą.
- **Node** - klasa reprezentująca pojedynczy węzeł lub liść drzewa.
- **Forest** - niedokończona implementacja lasu losowego złożonego z **n** drzew.
- **Attribute** - klasa reprezentująca pojedynczy atrybut i jego wartości.
- **Connection** - reprezentacja połączenia między dwoma węzłami (Node'ami), zawiera węzeł-parent, wartość atrybutu oraz węzeł-child.
- **main** - główny plik, w którym testujemy nasze implementacje. Zawiera też funkcje służące do wczytywania danych.

3.4 Opis algorytmów i zbiorów danych

3.4.1 Opis wykorzystanego algorytmu

Algorytm C5.0 na początku decyduje w jaki sposób powinien podzielić drzewo. Podczas decyzji tej kieruje się wartością entropii. Zestawy danych o wysokiej entropii będą bardzo zróżnicowane oraz będą dostarczać niewiele informacji ze względu na brak spójności. Zestawy danych o niskiej entropii będą mniej zróżnicowane i będą mogły dostarczać nam informacje odnośnie danych. Algorytm C5.0 stara się znaleźć taki podział, aby zmniejszyć entropię, czyli aby zwiększyć jednorodność w grupach danych. Dla n klas wartość entropii będzie mieścić się w przedziale od 0 do $\log_2(n)$. Matematycznie wzór na entropię przedstawia się następująco: $\sum_{i=1}^{c} -p_i log_2(p_i)$, gdzie c to ilość klas, a p_i to stosunek ilości danych z klasy i do wszystkich danych. Dany podział oceniamy pod względem

zysku informacyjnego, który obliczany jest jako różnica w entropii przed i po podziale. Jedną z komplikacji, która pojawia się w tym momencie jest to, iż dane dzielone są na więcej niż jedną partycję. W związku z tym, obliczając wartość entropii po powinniśmy brać pod uwagę całkowitą wartość entropii w każdej z powstałych partycji. Matematycznie można to zapisać następującym wzorem: $\sum_{i=1}^{n} w_i Entropy(P_i)$, gdzie n to ilość partycji, a w_i to stosunek ilości danych przypadających na daną partycję. Im wyższa jest wartość zysku informacji, tym lepszy jest dany atrybut w tworzeniu jednorodnych grup po podziale na ten atrybut. Maksymalny zysk informacyjny jest równy wartości entropii przed podziałem. W takiej sytuacji entropia po podziale wynosiłaby zero, co skutkuje podziałem na całkowicie jednorodne grupy.
Drzewo decyzyjne może rosnąć w nieskończoność, dopóki każdy przykład nie zostanie doskonale sklasyfikowany lub dopóki algorytmowi nie zabraknie funkcji do podziału. W sytuacji, gdy drzewo decyzyjne staje się zbyt duże, wiele podejmowanych decyzji będzie zbyt szczegółowych i model może nadmiernie dopasować się do danych uczących. Aby przeciwdziałać temu, stosuje się przycinanie drzewa decyzyjnego, które polega na zmniejszeniu jego rozmiaru, tak aby model dobrze klasyfikował dla każdych danych, nie tylko uczących, lub wczesne zatrzymanie algorytmu, jednak wadą tego rozwiązania jest to, iż nie wiemy, czy algorytm nie zatrzyma się akurat przed ważnymi danymi, dzięki którym nauczył by się lepiej. Ze względu na to stosować będziemy przycinanie - najpierw wyhodujemy drzewo, które będzie celowo duże, a następnie przytniemy węzły liści, aby zmniejszyć drzewo do odpowiedniego rozmiaru. Dzięki takiej kolejności mamy pewność, że wszystkie ważne struktury danych zostaną odkryte przed przycięciem.

Powyższy algorytm uzupełniliśmy również o obsługę danych ciągłych. Różnica między algorytmem dla danych dyskretnych a dla danych ciągłych polega na podziale danych. Wybierana jest mediana ze zbioru i według niej dzielone jest drzewo: na dane mniejsze od niej i dane większe bądź równe od niej. Algorytm rozpoznaje dane ciągłe od dyskretnych poprzez analizę pliku *adult.names*. Plik *.names* jest standardowym plikiem dołączonym do większości zbiorów danych. W pliku opisany jest każdy z atrybutów zbioru. Poniżej przedstawiamy jak wygląda część, dzięki której algorytm rozpoznaje dane dyskretne od danych ciągłych:

```
age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: continuous.
marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed,
Married-spouse-absent, Married-AF-spouse.
occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial,
Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing,
Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Female, Male.
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
```

3.4.2 Opis zbiorów danych

Do naszego projektu wybraliśmy dwa zbiory danych: *car evaluation* oraz *adult*. Poniżej znajdują się opisy obu zbiorów.

Car evaluation to zbiór danych oceny samochodów znajdujący się na tej stronie. W zbiorze znajduje się 1728 instancji danych. Nie występują w nim brakujące wartości.

Atrybuty:

- buying - wysokość ceny zakupy (vhigh, high, med, low),
- maint - wysokość ceny utrzymania (vhigh, high, med, low),
- doors - liczba drzwi (2, 3, 4, 5, more),
- persons - liczba osób (2, 4, more),
- lug_boot - rozmiar bagażnika (small, med, big),
- safety - szacowane bezpieczeństwo samochodu (low, med, high).

Poniżej przedstawiona jest ilość wystąpień danych klas oraz procentowy stosunek instancji danej klasy do ilości wszystkich danych.

```
class       N          N[%]
------------------------------
unacc     1210     (70.023 %)
acc        384     (22.222 %)
good        69     ( 3.993 %)
v-good      65     ( 3.762 %)
```

Adult to zbiór danych ze spisu ludności znajdujący się na tej stronie. W zbiorze znajduje się 48842 instancji danych. Występują w nim brakujące wartości - 7% instancji posiada brakujące wartości. Zbiór ten pozwala nam przewidywać, czy dochód danej osoby przekracza 50 000 USD rocznie.

Atrybuty:

- age - wiek osoby (atrybut ciągły),
- workclass - rodzaj zatrudnienia (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked),
- fnlwgt - charakterystyka demograficzna (atrybut ciągły),
- education - wykształcenie (Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool),
- education-num - (atrybut ciągły),
- marital-status - status cywilny (Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse),
- occupation - zawód (Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces),
- relationship - status związku (Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried),
- race - rasa (White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black),
- sex - płeć (Female, Male),
- capital-gain - zysk kapitałowy (atrybut ciągły),
- capital-loss - strata kapitałowa (atrybut ciągły),
- hours-per-week - liczba godzin pracy na tydzień (atrybut ciągły),
- native-country - kraj pochodzenia (United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, TrinadadTobago, Peru, Hong, Holand-Netherlands).

Poniżej przedstawiony jest procentowy stosunek instancji danej klasy do ilości wszystkich danych.

```
'>50K'  : 23.93% (z brakującymi wartościami) / 24.78% (bez brakujących wartości)
'<=50K' : 76.07% (z brakującymi wartościami) / 75.22% (bez brakujących wartości)
```

3.5 Ranking atrybutów

Ranking atrybutów wykonaliśmy na dwa różne sposoby:

3.5.1 Pierwszy sposób

Pierwszy sposób sporządzenia rankingu atrybutów opisaliśmy w dokumentacji wstępnej. Tworzony jest według następującej metryki: Tworzymy licznik "ważności" każdego z atrybutów i inicjalizujemy każdy zerem. W momencie algorytmu gdy atrybut jest wybierany przez *attribute selection method*, czyli przez minimalną entropię, dodajemy do licznika moc aktualnie rozważanego podzbioru. Opisane kroki wykonamy dla drzew zbudowanych na różnych podzbiorach S. W ten sposób liczniki atrybutów posortowane malejąco stworzą ranking mówiący o tym ile danych dzieli dany atrybut.
Opisany wyżej sposób realizowany jest w kodzie w następującym miejscu:

```
def attribute_ranking_by_count(data_file, names_file, no_trees=100):
    data = get_training_and_eval_sets(data_file, names_file, frac_arg=1)
    continuous_attributes = data[2]
    data = data[0]
    attributes_ranking = {}
    for column in data.columns:
        if column != 'class':
            attributes_ranking[column] = 0

    for i in range(no_trees):
        split_dataset = data.sample(frac=(1 / 10), random_state=np.random.RandomState())
        tree = Tree(split_dataset, continuous_attributes)
        tree.root = tree.generate_tree(tree.S, attributes_ranking)

    attributes_ranking = sorted(attributes_ranking.items(), key=lambda x: x[1], reverse=True)
    print(attributes_ranking)
```

Wyniki działania tej funkcji na dwóch rozważanych zbiorach danych przedstawiamy poniżej. Każdy wiersz przedstawia sumę rozpatrywanej metryki dla 1000 różnych drzew trenowanych na różnych (małych) podzbiorach zbioru trenującego:

Car dataset					
safety	persons	buying	maint	doors	lug_boot
159355	121644	63676	56215	20346	20750
158794	122296	63961	56238	20527	19801
159573	121071	63338	55632	21150	20014
159822	120756	63677	55428	20762	19847
157347	122871	63824	55282	21125	20839
158435	121624	63831	56090	20844	20535
157987	121967	64198	56092	20753	20346
158482	122991	63196	56007	20942	19976
158194	121946	64247	56424	20709	19849
159329	121717	63331	55681	21370	19880

Ranking jest posortowany malejąco od lewej do prawej. Wynika z niego, że w ocenie opłacalności kupna samochodu, nasze drzewa dużo bardziej kierują się atrybutami takimi jak bezpieczeństwo czy maksymalną liczba osób w samochodzie, niż liczbą drzwi czy pojemnością bagażnika. Należy zaznaczyć jednak, że istnieje powiązanie między tymi atrybutami. Można podejrzewać, że liczba osób może mówić też pośrednio o liczbie drzwi czy pakowności bagażnika. Z tego powodu nie można powiedzieć, że te dwa atrybuty (doors i lug_boot) nie są ważne w wyborze samochodu. Można za to stwierdzić, że są one znacząco mniej użyteczne dla drzew klasyfikacyjnych niż atrybuty znajdujące się wyżej w rankingu.

Adult dataset cz. 1						
relationship	education	occupation	workclass	capital-gain	age	marital-status
3207674	2865279	2490590	1314382	1092396	1065100	850820
3201840	2868180	2486026	1321672	1085570	1058655	858351
3218136	2856852	2496806	1322824	1098569	1073514	823797
3210793	2867519	2486055	1323725	1086150	1069085	838939
3228347	2862672	2498123	1320902	1076575	1065715	830570
3223042	2875904	2485605	1323270	1070304	1068714	820048
3230089	2871260	2492293	1319107	1082184	1071129	822697
3217718	2867648	2479836	1325665	1104970	1060236	842905
3202131	2874033	2493464	1324594	1084036	1064624	839122
3222288	2859366	2502288	1317869	1097764	1071672	830182

Adult dataset cz. 2						
hours-per-week	fnlwgt	native-country	race	capital-loss	sex	education-num
806611	779579	718796	597877	560561	387951	303260
808889	781004	726766	595585	561220	389417	302685
806522	784718	721626	599570	560116	387654	303618
815131	782648	719624	597100	559246	388734	301895
807449	783851	724544	593625	563642	389182	303108
802260	783843	727185	597769	564108	390892	302961
813765	784501	729623	596095	559286	389338	304262
815845	780779	715311	596213	557844	390881	302747
810821	780725	727634	596695	557762	386620	301040
803553	779813	726860	597752	559656	391559	302770

Wnioski wynikające z rankingu sporządzonego na drugim rozpatrywanym zestawie danych są jeszcze ciekawsze. Wynika z nich, że najważniejszą rolę w przewidywaniu zarobków osoby pełni ich status związku, jest on nawet ważniejszy niż wykształcenie danej osoby (które jest drugie w kolejności). Po drugiej stronie zestawienia znajduje się płeć oraz liczba lat edukacji. Niestety nie czujemy się wystarczająco wykształceni w kierunkach ekonomiczno-społecznych, żeby skomentować tak niskie położenie atrybutu związanego z płcią (ale cieszymy się z tego powodu). Możemy za to przypuszczać powód tak niskiego położenia liczby lat edukacji, przypuszczamy, że zawiera się on w innym atrybucie - "education" - i dlatego znajduje się tak nisko.

3.5.2 Drugi sposób

Na drugi sposób natrafiliśmy w trakcie uczenia się do kolokwium i przeglądania książki napisanej przez Pana Profesora Pawła Cichosza - "Data Mining Algorithms" - w rozdziale o lasach losowych znajduje się zapis o skutkach ubocznych lasów. "Attribute utility Out-of-bag instances are useful not only for estimating model quality, but also (for estimating) attribute utility, which may be viewed as particular attributes' impact on the former" (strona 446). Dlatego, zgodnie z metodą przedstawioną w książce sporządziliśmy rankingi atrybutów na drugi sposób. Uczyliśmy wiele drzew na małych podzbiorach danych i przy testowaniu wydajności mieszaliśmy wartości pojedynczego atrybutu. W ten sposób można sprawdzić jak zmiana pojedynczego atrybutu wpływa na jakość predykcji drzew. Do przetestowania każdego atrybutu wykorzystywaliśmy 1000 drzew i wyciągaliśmy średnią z precyzji ich predykcji. Opisaną funkcję realizuje poniższy kod:

```
1  def attribute_ranking_by_attribute_poisoning(data_file, names_file,
   ↪  no_trees=100):
2      data = get_training_and_eval_sets(data_file, names_file, frac_arg=(8 / 10))
3      continuous_attributes = data[2]
4      train_data = data[0]
5      test_data = data[1]
6      attributes_ranking = {}
7      trained_models = []
8      # train no_trees models
9      for model_index in range(no_trees):
10         split_dataset = train_data.sample(frac=(1 /
           ↪  10),random_state=np.random.RandomState())
11         tree = Tree(split_dataset, continuous_attributes)
12         tree.root = tree.generate_tree(tree.S)
13         trained_models.append(tree)
14
15     # check on control sample
16     control_sample_results = []
17     for tree in trained_models:
18         control_sample_results.append(get_prediction_results(tree, test_data,
           ↪  if_print=False))
19     attributes_ranking['control'] = mean(control_sample_results)
20     for column in train_data.columns:
21         if column != 'class':
22             poisoned_test_df = randomly_permute_column(test_data, column)
23             attribute_poison_results = []
24             for tree in trained_models:
25                 attribute_poison_results.append(get_prediction_results(tree,
                   ↪  poisoned_test_df, if_print=False))
26             attributes_ranking[column] = mean(attribute_poison_results)
```

```
27
28      attributes_ranking = sorted(attributes_ranking.items(), key=lambda x: x[1],
        ↪  reverse=True)
29      print(attributes_ranking)
```

Poniżej przedstawiamy wyniki otrzymane po po 10-krotnym przetestowaniu na 1000 drzew i na obu zbiorach danych. Zaznaczyć należy, że tutaj im mniejsza liczba tym atrybut jest ważniejszy (odwrotnie niż poprzednia metryka). Pierwsza kolumna "control" pokazuje wyniki z próbki testowej - gdy nie zmienimy żadnych atrybutów.

Car dataset						
control	doors	lug_boot	maint	buying	persons	safety
0.8181	0.8180	0.8060	0.7696	0.7639	0.7074	0.6741
0.7903	0.7910	0.7813	0.7553	0.7509	0.6788	0.6456
0.7748	0.7749	0.7659	0.7384	0.7174	0.6429	0.6302
0.7755	0.7756	0.7668	0.7400	0.7238	0.6692	0.6218
0.8049	0.8064	0.7976	0.7696	0.7571	0.6727	0.6760
0.7969	0.7982	0.7873	0.7577	0.7490	0.6918	0.6383
0.7985	0.7973	0.7899	0.7530	0.7462	0.6740	0.6411
0.7782	0.7796	0.7708	0.7362	0.7441	0.6876	0.6366
0.7837	0.7823	0.7737	0.7536	0.7322	0.6689	0.6489
0.7786	0.7817	0.7666	0.7411	0.7194	0.6824	0.6566

W przypadku atrybutów mówiących o samochodach ranking nie uległ zmianie względem pierwszej zastosowanej metody. Zaznaczyć jednak należy, że otrzymaliśmy lepszą estymację informacji zawartych w atrybutach. Można na przykład zauważyć że w paru przypadkach zmiana atrybutu "doors" spowodowała poprawienie precyzji predykcji - pokazuje to, że atrybut ten jest prawie bezużyteczny.

Adult dataset cz. 1							
control	education-num	native-country	fnlwgt	race	capital-loss	sex	hours-per-week
0.7800	0.7800	0.7798	0.7796	0.7794	0.7784	0.7782	0.7765
0.7855	0.7855	0.7856	0.7854	0.7853	0.7842	0.7840	0.7833
0.7840	0.7840	0.7839	0.7836	0.7836	0.7825	0.7823	0.7815
0.7827	0.7827	0.7826	0.7827	0.7827	0.7813	0.7813	0.7803
0.7868	0.7868	0.7867	0.7871	0.7867	0.7856	0.7849	0.7848
0.7830	0.7830	0.7828	0.7827	0.7826	0.7807	0.7812	0.7808
0.7789	0.7789	0.7788	0.7791	0.7786	0.7779	0.7769	0.7765
0.7837	0.7837	0.7838	0.7828	0.7832	0.7831	0.7820	0.7830
0.7839	0.7840	0.7840	0.7835	0.7839	0.7821	0.7820	0.7824
0.7887	0.7887	0.7886	0.7888	0.7885	0.7872	0.7872	0.7864

Adult dataset cz. 2						
workclass	age	marital-status	capital-gain	occupation	education	relationship
0.7770	0.7753	0.7758	0.7721	0.7654	0.7391	0.7158
0.7822	0.7810	0.7797	0.7777	0.7717	0.7486	0.7242
0.7818	0.7799	0.7808	0.7768	0.7677	0.7431	0.7209
0.7792	0.7778	0.7762	0.7756	0.7677	0.7470	0.7192
0.7827	0.7824	0.7790	0.7791	0.7695	0.7467	0.7285
0.7788	0.7788	0.7757	0.7756	0.7686	0.7432	0.7135
0.7749	0.7736	0.7734	0.7702	0.7640	0.7435	0.7163
0.7793	0.7791	0.7792	0.7772	0.7698	0.7449	0.7245
0.7805	0.7812	0.7774	0.7766	0.7687	0.7442	0.7238
0.7846	0.7856	0.7854	0.7812	0.7700	0.7458	0.7261

W rankingu atrybutów zbioru mówiącego o zarobkach zmieniło się trochę więcej. Kolejność najważniejszych atrybutów pozostała taka sama ale na przykład okazało się, że atrybut mówiący o płci ma więcej znaczenia niż sugerowały poprzednie wyniki. Zaobserwować również można, że większość wyników jest bardzo podobna i bliska próbie kontrolnej. Może to sugerować, że zaburzenie wartości pojedynczego atrybutu wpływa w znacznie mniejszym stopniu na jakość predykcji (niż w przypadku poprzedniego zbioru danych). Drzewa potrafią przewidzieć klasy próbek na poziomie bliskim do próbki kontrolnej nawet dla zbiorów danych o zaburzonych wartościach pojedynczego atrybutu. Może to na przykład wynikać z znacznie większej liczby podawanych atrybutów niż w przypadku zbioru danych samochodów.

W tej sekcji należy zaznaczyć, że precyzja predykcji (nawet dla próby kontrolnej), nie była za duża. Wynika to z bardzo małych zbiorów uczących. Wykorzystując normalne proporcje wielkości zbiorów trenujących do testujących osiągane są znacząco wyższe wyniki.

3.6 Inne testy i eksperymenty

3.6.1 Jakość predykcji

Poniżej ponownie przedstawiamy dwa przykładowe wyniki wraz z ich interpretacją:

Wyniki dla zbioru Car z podziałem danych uczących/testujących w stosunku (4:3)

```
Correct = 687 | Incorrect = 54 | Overall_accuracy = 0.9271255060728745
    ========= unacc =========
        TP = 498 | FN = 24 | FP = 11 | TN = 189
        TPR = 0.9540229885057471
        TNR = 0.945
        PPV = 0.9783889980353635
        ACC = 0.9515235457063712
        F1 = 0.9660523763336566
    ========= acc =========
        TP = 147 | FN = 23 | FP = 23 | TN = 540
        TPR = 0.8647058823529412
        TNR = 0.9591474245115453
        PPV = 0.8647058823529412
        ACC = 0.937244201909959
        F1 = 0.8647058823529412
    ========= vgood =========
        TP = 24 | FN = 2 | FP = 9 | TN = 663
        TPR = 0.9230769230769231
        TNR = 0.9866071428571429
        PPV = 0.7272727272727273
        ACC = 0.9842406876790831
        F1 = 0.8135593220338984
    ========= good =========
        TP = 18 | FN = 5 | FP = 11 | TN = 669
        TPR = 0.782608695652174
        TNR = 0.9838235294117647
        PPV = 0.6206896551724138
        ACC = 0.9772403982930299
        F1 = 0.6923076923076923
 Error = 0
============ FINISHED =============
```

Jak widać powyżej, otrzymany wynik to prawie 93% precyzji. Wynik taki otrzymany został na wielkości zbioru testowego 4/7 całości danych. Przetestowaliśmy inne proporcje ale zwiększanie rozmiaru danych trenujących nie zwiększało już precyzji predykcji. Używając podanej proporcji danych trenujących do testowych uzyskiwaliśmy średnio 90% precyzji.

Oprócz obliczenia ogólnej precyzji predykcji, policzyliśmy więcej metryk dla predykcji każdej z klas. W wynikach szczegółowych można zaobserwować, że metryki predykcji dla poszczególnych klas znacząco różnią się dla różnych klas. Dla klasy **"unacc"**, metryki są najlepsze, dla klasy **"acc"** metryki są trochę gorsze, dla **"vgood"** jeszcze gorsze, a dla klasy **"good"** najgorsze. Prawdopodobnie wynika to z zawartości próbek o danych klasach w używanym przez nas zbiorze:

```
class      N          N[%]
----------------------------
unacc     1210     (70.023 %)
acc        384     (22.222 %)
good        69     ( 3.993 %)
v-good      65     ( 3.762 %)
```

Próbek o klasie **"unacc"** jest najwięcej więc najłatwiej tę klasę przewidzieć, analogiczna sytuacja występuje dla klasy **"acc"**. Ciekawe jest to, że klasa **"vgood"** jest przewidywana lepiej (biorąc

na przykład pod uwagę metrykę F1) niż klasa **"good"** mimo mniejszej liczby przykładów trenujących. Może to na przykład wynikać z większych różnic w wartościach atrybutów.

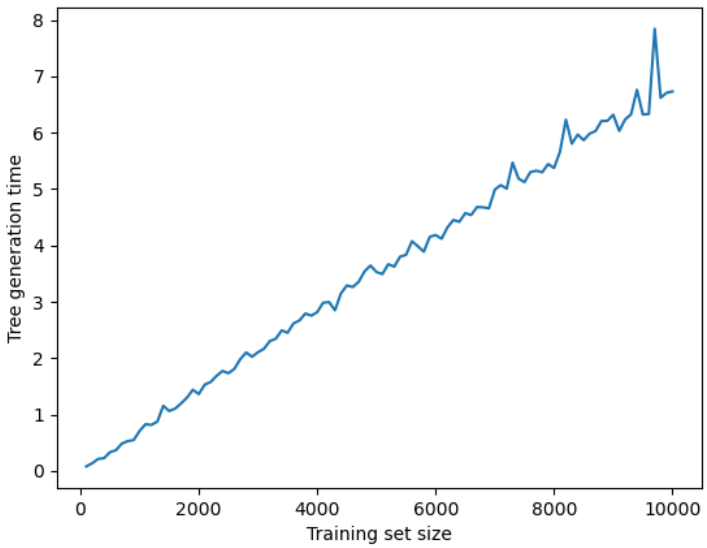
Wyniki dla zbioru Adult z podziałem danych uczących/testujących w stosunku (4:1)

```
Correct = 5291 | Incorrect = 1221 | Overall_accuracy = 0.8125
    =========  >50K =========
        TP = 926 | FN = 705 | FP = 516 | TN = 4365
        TPR = 0.5677498467198038
        TNR = 0.8942839582052858
        PPV = 0.6421636615811374
        ACC = 0.8125
        F1 = 0.6026684022128214
    =========  <=50K =========
        TP = 4365 | FN = 516 | FP = 705 | TN = 926
        TPR = 0.8942839582052858
        TNR = 0.5677498467198038
        PPV = 0.8609467455621301
        ACC = 0.8125
        F1 = 0.8772987639433223
 Error = 0
============= FINISHED =============
```

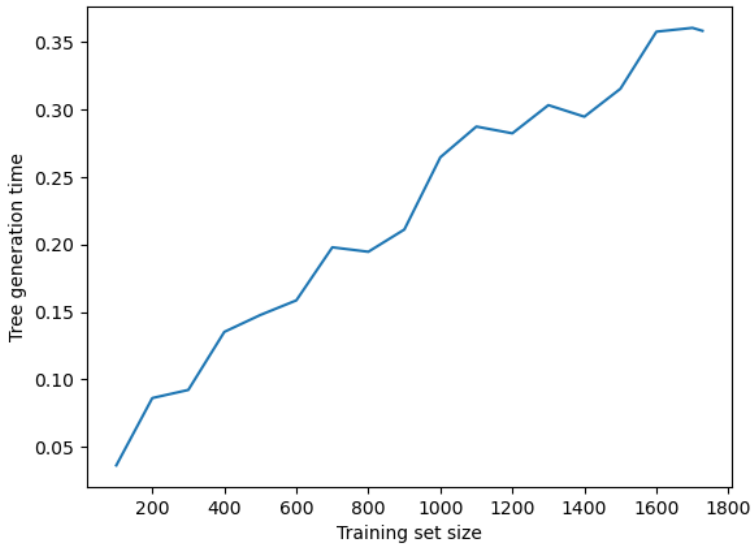
Powyżej widać, że udało się nam uzyskać precyzję jedynie około 81%. Średnia precyzja, która wychodziła w testach wynosiła około 79%. Precyzja to może wynikać z uzupełniania brakujących danych, ale odrzucenie przykładów zawierających brakujące dane podniosło precyzję jedynie o jeden punkt procentowy. Dla tego zbioru nie trzeba było obliczać metryk dla obu klas, jednakże nasz program zrobił to automatycznie. Tutaj ponownie widać wpływ zawartości danych klas w wybranym zbiorze - przykładów o klasie **"¡=50k"** jest znacząco więcej niż drugiej klasy.

3.6.2 Testy wydajnościowe

W ramach innych testów i eksperymentów postanowiliśmy obliczyć czas generowania się drzewa dla obu zbiorów oraz czas predykcji 100 przykładów testowych w zależności od ilości danych trenujących. Testy zostały przeprowadzone na laptopie z systemem Windows 10, z procesorem R7-4800H i używając condy 3.9 jako dystrybucji pythona. Poniżej znajdują się wykresy stworzone w ramach testów, na pionowej osi znajduje się czas w sekundach a na poziomej rozmiar zbioru:

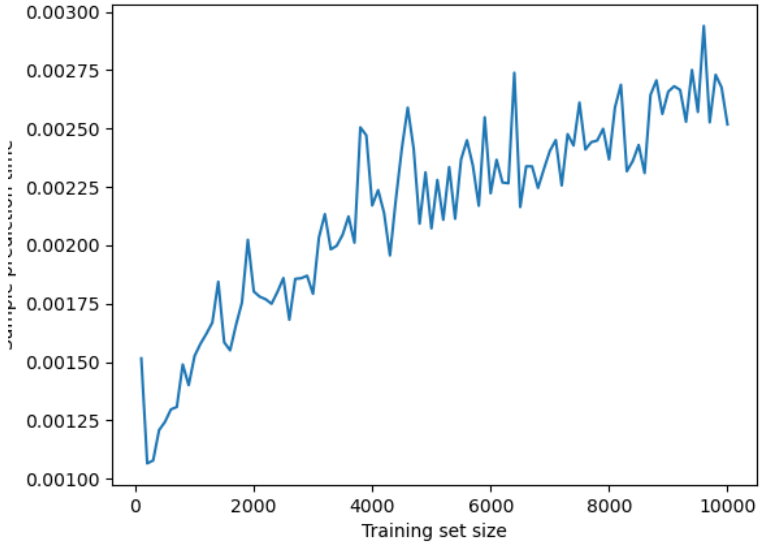


Rysunek 2: Czas tworzenie się drzewa dla zbioru danych *adult*

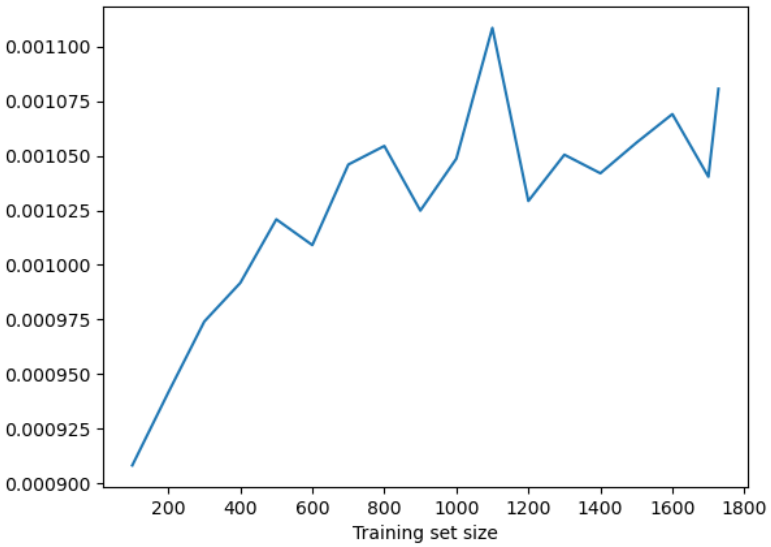


Rysunek 3: Czas tworzenie się drzewa dla zbioru danych *car*

Na powyższych wykresach można zauważyć, iż czas tworzenia się drzewa w zależności od rozmiaru zbioru danych trenujących jest praktycznie liniowy.



Rysunek 4: Czas predykcji 100 przykładów testowych dla zbioru danych *adult*



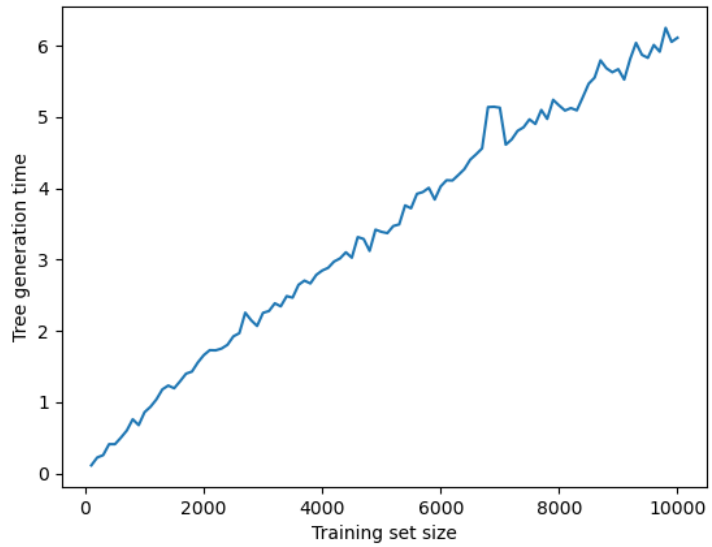
Rysunek 5: Czas predykcji 100 przykładów testowych dla zbioru danych *car*

Na dwóch powyższych wykresach można zaobserwować, iż czas predykcji 100 przykładów testowych jest zależny logarytmicznie od rozmiaru zbioru danych trenujących (a przynajmniej przypominają zależność logarytmiczną). Na wykresach można zaobserwować dużo skoków. Mogą one wynikają one z faktu, iż drzewo nie jest pełne, przez co niektóre wyniki mogą zostać przewidziane szybciej niż inne. Drugim powodem tych skoków może być precyzja samego zegara zliczającego czas wykonania funkcji - w naszej architekturze precyzja ta wynosi aż około 16 milisekund co jest porównywalne z czasem predykcji 100 przykładów.

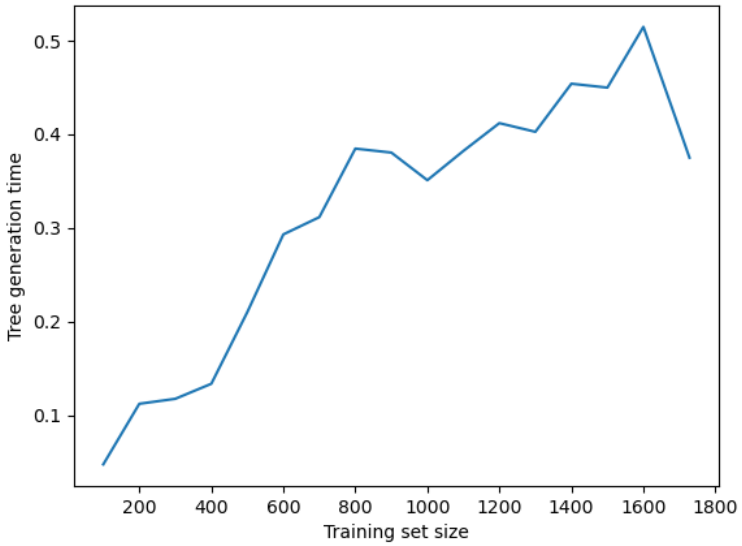
Ponieważ podejrzewaliśmy, że największym obciążeniem obliczeniowym i czasowym naszego algorytmu jest obliczanie entropii w **attribute_selection_method**, to postanowiliśmy sprawdzić to podejrzenie. Przetestowaliśmy nasz algorytm również dla uproszczonej funkcji liczenia entropii, która nie bierze pod uwagę wartości klas danych przykładów, a jedynie to jak różne są wartości danego atrybutu korzystając z entropii Shanonna. Powyższą funkcjonalność realizuje następująca funkcja:

```
1      def attribute_selection_method(self, S):
2          entropy_list = []
3          for column in S:
4              if column != 'class':
5                  entropy_list.append(entropy(S[column].values))
6          return entropy_list.index(min(entropy_list))
```

Gdy zmodyfikowaliśmy tę funkcję przetestowaliśmy naszą implementację jeszcze raz, poniżej przedstawiamy wyniki tych testów:

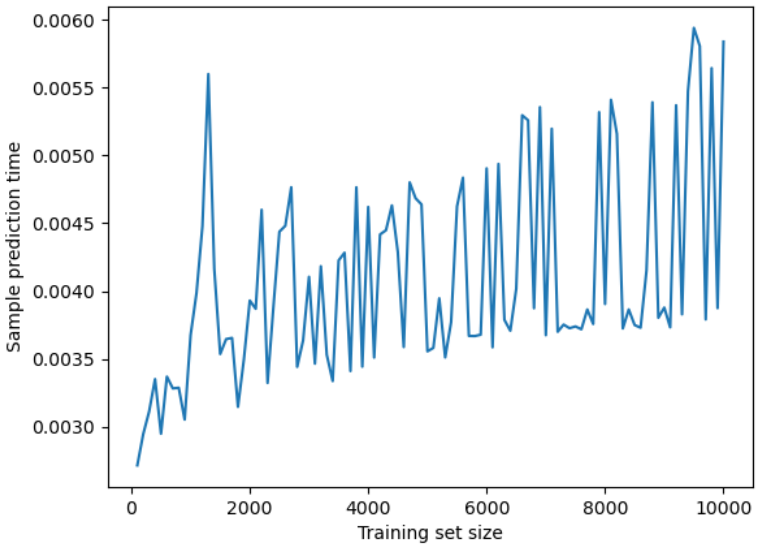


Rysunek 6: Czas tworzenie się drzewa dla zbioru danych *adult* z entropią wyliczaną przy pomocy zmodyfikowanej funkcji

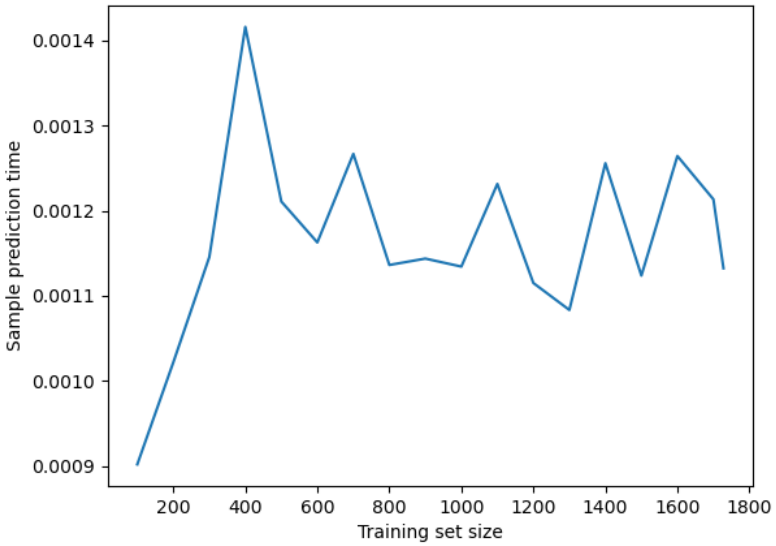


Rysunek 7: Czas tworzenie się drzewa dla zbioru danych *car* z entropią wyliczaną przy pomocy zmodyfikowanej funkcji

Porównując powyższe wykresy do poprzednich wykresów, można zauważyć, iż czas tworzenia się drzewa dla zbioru *adult* jest mniejszy. Różnica wynosi 2 sekundy, co dla całego czasu równego wcześniej 8 jest znaczącą zmianą. Natomiast dla zbioru danych *car* czas tworzenia jest nieco dłuższy o około 0.10-0.15 sekundy.



Rysunek 8: Czas predykcji 100 przykładów testowych dla zbioru danych *adult* z entropią wyliczaną przy pomocy zmodyfikowanej funkcji



Rysunek 9: Czas predykcji 100 przykładów testowych dla zbioru danych *car* z entropią wyliczaną przy pomocy zmodyfikowanej funkcji

Porównując czasy predykcji 100 przykładów testowych, można zauważyć, iż czas predykcji dla zbioru *adult* z entropią przy pomocy zmodyfikowanej funkcji jest 2 razy większy niż czas z entropią liczoną zgodnie ze sposobem podanym na wykładzie. Jeśli chodzi o czas predykcji dla zbioru *car* z entropią liczoną przy pomocy zmodyfikowanej funkcji jest on porównywalny do czasu predykcji z entropią liczoną zgodnie ze sposobem podanym na wykładzie. Takie różnice prawdopodobnie wynikają z mniej optymalnych podziałów, jakie drzewa wygenerowane z uproszczoną funkcją, generują. Precyzja predykcji tych drzew także się zmienia, dla zbioru *Car* spada nawet o osiem punktów procentowych, natomiast dla zbioru *adult* precyzja pozostaje mniej więcej taka sama.

4 Podsumowanie

Projekt był bardzo przyjemny do realizacji i uważamy, że dużo z niego wynieśliśmy. Przekonał nas do używania drzew klasyfikacyjnych oraz lasów losowych do przewidywania wyników i zachęcił do używania ich w kolejnych projektach.