Politechnika Białostocka

Sztuczna inteligencja

Sprawozdanie z projektu końcowego

Temat: Prezentacja możliwości biblioteki scikit-learn w projektach informatycznych, wymagających użycia drzew decyzyjnych

Wykonujący projekt: **Adam Bajguz**
Magdalena Kalisz

Studia dzienne
Kierunek: Informatyka
Semestr: IV Grupa zajęciowa: **PS1**
Prowadzący ćwiczenie: **mgr inż. Dariusz Jankowski**

12 czerwca 2018
Data wykonania projektu

.....
Data i podpis prowadzącego

Spis treści

1 Wstęp **2**
1.1 Drzewa decyzyjne w teorii decyzji i uczeniu maszynowym . . . 2
1.2 Rodzaje drzew decyzyjnych . . . 2

2 Budowa drzew decyzyjnych **2**

3 Idea drzew decyzji **3**

4 Cechy drzew decyzyjnych **3**

5 Rodzaje kryteriów podziału **4**
5.1 Współczynnik Giniego . . . 4
5.2 Information gain . . . 4
5.3 Gain ratio . . . 4

6 Algorytmy budowania drzew decyzyjnych **4**
6.1 Algorytm CART . . . 5
6.2 Algorytm CHAID . . . 5
6.3 Algorytm ID3 . . . 5
6.4 Algorytm C4.5 . . . 6
6.5 Algorytm C5.0 . . . 6

7 Przycinanie drzew decyzyjnych **6**

8 Biblioteka scikit-learn **6**
8.1 Wstęp . . . 6
8.2 Moduł drzew decyzyjnych . . . 7
8.2.1 Klasa DecisionTreeClassifier . . . 7
8.2.2 Klasa DecisionTreeRegressor . . . 8

9 Opis wykorzystanych zbiorów danych **9**
9.1 Zbiór danych Forest Fires . . . 10
9.2 Zbiór danych Mushroom . . . 10

10 Opis programów i wyników **11**
10.1 Zbiór danych Forest Fires . . . 11
10.2 Zbiór danych Mushroom . . . 12

11 Wnioski **40**

12 Podział pracy **40**

Literatura **41**

1 Wstęp

Drzewa decyzyjne to nieparametryczna nadzorowana metoda uczenia, która może być stosowana zarówno do klasyfikacji, jak i regresji. Pierwszy artykuł, który przedstawia podejście do klasyfikacji znane z drzew decyzyjnych, pochodzi z 1959 roku [1]. Z kolei pierwszy algorytm drzew regresyjnych został opublikowany w roku 1963 [2].

Drzewa decyzyjne są jednym z najczęściej wykorzystywanych narzędzi do klasyfikacji danych i prognozowania z uwagi na fakt, że wiedza odkryta przez drzewo decyzyjne jest zilustrowana w hierarchicznej strukturze. Technika drzew decyzyjnych, czy też klasyfikacyjnych, pozwala m.in. na [3, 4]:

- wyznaczenie zasad decyzyjnych opisujących reguły przypisywania obiektów do wyróżnionych klas (zasady odwołują się do wartości atrybutów opisujących obiekty),
- analizę zbioru obiektów opisywanych przez przyjęty zestaw atrybutów, której celem jest doskonalenie podziału obiektów na jednorodne klasy.

1.1 Drzewa decyzyjne w teorii decyzji i uczeniu maszynowym

Drzewa decyzyjne to również graficzny sposób wspierania procesu decyzyjnego. Drzewo stosowane jest w teorii decyzji i ma szereg zastosowań. Może zarówno rozwiązać problem decyzyjny, jak i stworzyć plan. Metoda drzew decyzyjnych sprawdza się również w momencie rozwiązywania problemów decyzyjnych z wieloma rozgałęziającymi się wariantami oraz podejmowania decyzji w warunkach ryzyka. Drzewa znalazły zastosowanie w takich dziedzinach jak botanika i medycyna, a nawet ekonomia, gdyż są w stanie ułatwiać i usprawniać komputerowe wspomaganie procesu podejmowania decyzji [2, 5].

1.2 Rodzaje drzew decyzyjnych

Rodzaj drzewa decyzyjnego zależy od typu zmiennej docelowej:

- drzewo decyzyjne zmiennych jakościowych (drzewo decyzyjne o zmiennej kategorialnej) - drzewo decyzyjne, które ma kategoryczną zmienną docelową,
- drzewo decyzyjne zmiennych ilościowych (ciągłych) - drzewo decyzyjne ma ciągłą (ilościową / numeryczną) zmienną docelową.

Zaletą drzew decyzyjnych jest to, że mogą modelować dowolny typ funkcji do klasyfikacji lub regresji. Do rozwiązywania problemów zarówno regresyjnych, jak i klasyfikacyjnych służy metoda CART (rozdział 6.1). Wyróżnia się dwa typy drzew decyzyjnych [6]:

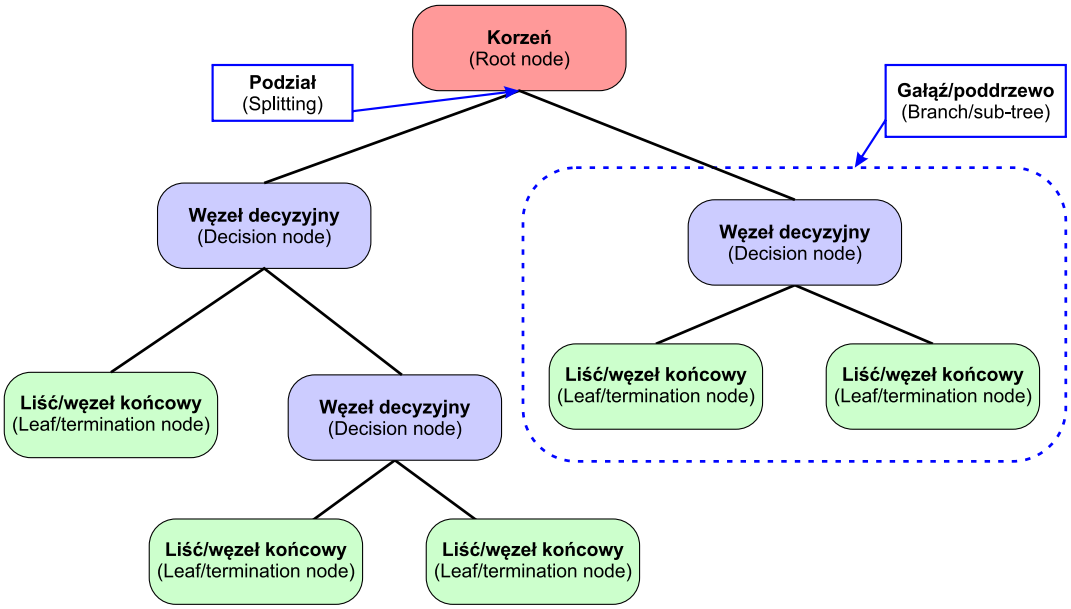
- drzewa klasyfikacyjne - służą porządkowaniu klas, charakteryzuje je kategoryczna zmienna zależna, której wartość (czyli przynależność przypadku do klasy, grupy) chcemy poznać na podstawie znajomości wartości jednej lub większej liczby predykcyjnych zmiennych ciągłych oraz, ewentualnie zmiennych kategorialnych;
- drzewa regresyjne - służą do przewidywania wartości zmiennej ciągłej, na podstawie znajomości wartości jednej lub większej liczby predykcyjnych zmiennych ciągłych.

2 Budowa drzew decyzyjnych

Drzewo decyzyjne buduje modele klasyfikacji l ub r egresji w p ostaci s truktury d rzewa, r ozkłada zbiór danych na coraz mniejsze podzbiory. Oznacza to, że drzewem decyzyjnym jest 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 (Rys. 1).

Drzewa decyzyjne charakteryzują się strukturą hierarchiczną (Rys. 1). Oznacza to, że w kolejnych krokach zbiór obiektów jest dzielony, poprzez odpowiedzi na pytania o wartości wybranych cech lub ich kombinacji liniowych. 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 [7, 8].



Rys. 1: Budowa (struktura) drzewa decyzyjnego

3 Idea drzew decyzji

Idea drzew opiera się na rekursywnym podziale danych na coraz to mniejsze sterty w celu jak najlepszego dopasowania. Drzewa decyzyjne kodują zestaw reguł if-else, które mogą być używane do przewidywania zmiennej docelowej danych funkcji danych. Reguły if-else są tworzone przy użyciu zestawu danych treningowych w celu zaspokojenia jak największej liczby instancji danych treningowych. Początkowo próbka (węzeł macierzysty, korzeń) dzielona jest na dwa lub więcej podzbiorów (węzły potomne). Natomiast węzeł optymalny wyszukuje się na podstawie wszystkich punktów węzłowych dla każdej zmiennej. Następnie proces jest powtarzany dla każdego węzła potomnego, a te podczas dzielenia traktowane są jak węzły macierzyste. Węzeł, którego nie można już podzielić nazywamy liściem, bądź węzłem końcowym, a liczbę liści - wielkością drzewa [3, 4, 6, 7, 9].

4 Cechy drzew decyzyjnych

Drzewa decyzyjne i uczenie się drzewa decyzyjnego razem stanowią prosty i szybki sposób uczenia się funkcji, która mapuje dane x na wyniki y, gdzie x może być mieszanką zmiennych jakościowych i liczbowych, a y może być kategoryczne dla klasyfikacji lub numeryczne dla regresji [6].

Największą zaletą drzew decyzyjnych jest to, że mogą modelować dowolny typ funkcji do klasyfikacji lub regresji, czego inne techniki nie potrafią. Ponadto drzewa decyzyjne są uważane za metodę nieparametryczną, co oznacza, że nie mają one żadnych założeń co do rozkładu danych i struktury klasyfikatora. Drzewa decyzyjne zapewniają również znacznie szybsze trenowanie w porównaniu z prostymi sieciami neuronowymi przy porównywalnej wydajności (złożoność czasowa drzew decyzyjnych jest funkcją zależną od liczby cech oraz wierszy w zestawie danych, podczas gdy dla sieci neuronowych jest funkcją zależną od liczby cech, wierszy w zestawie danych, warstw ukrytych oraz węzłów w każdej ukrytej warstwie). Wadą drzew decyzyjnych jest podatność na przeuczenie. Zatem drzewa decyzyjne są używane najczęściej w przypadku bardzo dużych zbiorów danych, które są uważane za dobrze reprezentujące rzeczywistość. Niektóre algorytmy przycinania drzew są używane do rozwiązania problemu przeuczenia [3, 6, 8, 10].

5 Rodzaje kryteriów podziału

W procecie budowy drzewa decyzyjnego należy wielokrotnie dokonać podziału zbioru danych, tj. należy zadać więcej niż jedno pytanie: co jest cechą, od której powinniśmy zacząć (węzeł główny) i w jakiej kolejności powinniśmy budować węzły wewnętrzne, to znaczy używać opisowych cech, aby podzielić zbiór danych? W związku z tym przydatne byłoby zmierzenie "informatywności" funkcji i wykorzystanie tej funkcji z największą "informacyjnością" jako cechą, która powinna być używana do dzielenia danych.

5.1 Współczynnik Giniego

Współczynnik Giniego lub indeks Giniego to miara koncentracji (nierównomierności) rozkładu zmiennej losowej. Nazwa współczynnika pochodzi od nazwiska jego twórcy, włoskiego statystyka Corrado Giniego. Jest on wykorzystywany przez algorytm CART do mierzenia tego, jak często losowo wybrany element z zestawu byłby niewłaściwie oznaczany, gdyby był losowo oznaczony zgodnie z rozkładem etykiet w podzbiorze. Gini index można obliczyć, sumując prawdopodobieństwo p_i elementu o etykiecie i wybieranej razy prawdopodobieństwo $\sum_{k \neq i} p_k = 1 - p_i$ o pomyłce w kategoryzacji tego przedmiotu. Osiąga minimum (zero), gdy wszystkie przypadki w węźle należą do jednej kategorii docelowe [8, 11].

5.2 Information gain

Information gain (wzmocnienie informacji) - atrybutem podziału jest ten, który ma maksymalny przyrost informacji. Aby móc obliczyć przyrost informacji, konieczne jest wprowadzenie terminu entropii zbioru danych. Entropia zbioru danych służy do pomiaru nierównomierności zbioru danych, a w przypadku drzew decyzyjnych jest używana jako miernik informatywności. Termin entropia (w teorii informacji) pochodzi od Claude'a E. Shannona. Idea entropii jest w uproszczeniu następująca: wyobraź sobie, że masz pudełko, które zawiera 100 białych kulek. Zestaw kulek w pudełku można uznać za całkowicie równomierny, ponieważ zawieraja tylko białe kulki (zbiór kulek ma entropię 0, tj. zero nierównomierności). Jeżeli 30 z tych kulek zostało by zastąpionych przez szare, a 20 przez czarne, to gdyby teraz zechcieć wyjąć jedną kulkę z pudełka to prawdopodobieństwo otrzymania białej kulki spadło by z 1,0 do 0,5. Oznacza to, że nierównomierność wzrosła, równomierność zmniejszyła się, a entropia wzrosła. Podsumowując, im bardziej nierównomierny jest zbiór danych, tym wyższa entropia, a im mniej nierównomierny zbiór danych, tym niższa entropia [3, 8]

5.3 Gain ratio

Gain ratio (współczynnik wzmocnienia) - wybiera atrybut o najwyższym wzroście informacji do liczby współczynników wartości wejściowych. Liczba wartości wejściowych to liczba odrębnych wartości atrybutu występującego w zbiorze treningowym. Jest on stosunkiem przyrostu informacji do wewnętrznej informacji. Jest stosowany w celu zmniejszenia błędu w kierunku atrybutów o wielu wartościach, biorąc pod uwagę liczbę i rozmiar oddziałów przy wyborze atrybutu [8, 11, 12].

6 Algorytmy budowania drzew decyzyjnych

Dostępnych jest kilka algorytmów służących do klasyfikacji i analizy segmentacji. Wszystkie te algorytmy zasadniczo realizują to samo zadanie: dzieląc dane na kolejne podgrupy, analizują wszystkie zmienne w zbiorze danych, by znaleźć zmienną zapewniającą najlepszą klasyfikację lub predykcję. Proces jest rekursywny, a grupy są dzielone na coraz mniejsze jednostki aż do ukończenia drzewa (zgodnie z określonym kryterium zatrzymania). Popularne algorytmy dzielenia obejmują minimalizację Gini Impurity (używanego przez CART) lub maksymalizację Information Gain (używanego przez ID3, C4.5) [13].

Przegląd istniejącej literatury pokazuje, że do najczęściej stosowanych algorytmów drzewa decyzyjnego należą algorytm Iterative Dichotomiser 3 (ID3), algorytm C4.5, CHAID oraz CART algorytm. Wśród tych algorytmów są pewne różnice, z których jedną jest możliwość modelowania różnych typów danych. Ponieważ zestaw danych może być skonstruowany na podstawie różnych typów danych, np. danych kategorycznych, danych numerycznych lub kombinacji obu, istnieje potrzeba użycia odpowiedniego algorytmu drzewa decyzyjnego, który może obsługiwać określony typ danych wykorzystywanych w zestawie danych. Wszystkie wyżej wymienione algorytmy mogą wspierać modelowanie danych jakościowych, podczas gdy tylko algorytm C4.5, C5.0 i algorytm CART mogą być używane do modelowania danych numerycznych

(Tab. 1). Inną różnicą między tymi algorytmami jest proces opracowywania modeli, szczególnie na eta-pie budowania i przycinania drzew. Algorytmy ID3, C4.5 i C5.0 dzielą model drzewa na tyle rozgałęzień ile można osiągnąć w danym momencie, podczas gdy algorytm CART obsługuje jedynie binarne podziały. Z kolei mechanizmy przycinania zlokalizowane w algorytmach C4.5, C5.0 i CART wspierają usuwanie nieistotnych węzłów i rozgałęzień. Natomiast algorytm CHAID nie wymaga dodatkowego etapu przycinania drzewa, z uwagi na fakt że CHAID stara się zapobiegać przeuczeniu od samego początku, wykorzystując pomysł wstępnego przycinania (węzeł jest dzielony tylko wtedy, gdy spełnione jest kryterium istotności) [8, 13].

Tab. 1: Porównanie popularnych algorytmów budowania drzew decyzyjnych

Algorytm	**Typ danych**	**Metoda podziału danych numerycznych**
CHAID [14]	Kategoryczne	Nie dotyczy
ID3 [4]	Kategoryczne	Nie dotyczy
C4.5 [15]	Kategoryczne, numeryczne	Brak ograniczeń
C5.0	Kategoryczne, numeryczne	Brak ograniczeń
CART [6]	Kategoryczne, numeryczne	Podziały binarne

6.1 Algorytm CART

Algorytm CART (ang. Classification and Regression Trees) jest popularnym algorytmem uczenia drzew decyzyjnych. W odróżnieniu od ID3 i C4.5, drzewo uczenia się w tym przypadku może być używane zarówno do klasyfikacji wieloklasowej, jak i do regresji w zależności od rodzaju zmiennej zależnej. Proces budowy drzewa składa się z rekurencyjnego dwójkowego podziału węzłów. Aby znaleźć najlepszy podział w każdym węźle, rozważane są wszystkie możliwe podziały wszystkich dostępnych atrybutów predykcyjnych. Najlepszy podział to taki, który maksymalizuje pewne kryterium podziału. Do zadań klasyfikacyjnych, tj. gdy atrybut zależny jest kategoryczny, jako kryterium podziału stosuje się indeks Giniego. Do zadań regresyjnych, tj. gdy zmienna zależna jest ciągła, stosowana jest metoda najmniejszych kwadratów [3, 6, 12].

6.2 Algorytm CHAID

CHAID to algorytm do uczenia drzew decyzyjnych zaproponowany przez Kassa (1980). Działa podobnie do CART - oba mogą być używane zarówno do klasyfikacji, jak i regresji. W przeciwieństwie do CART, CHAID wewnętrznie obsługuje tylko kategoryczne funkcje. Funkcje ciągłe są najpierw konwertowane na zmienne kategoryczne za pomocą grupowania/kubełkowania (ang. binning). Liczba kubełków (ang. bins) (K) musi być dostarczona przez użytkownika. Biorąc pod uwagę K, predykator jest podzielony w taki sposób, że wszystkie kubełki mają mniej więcej taką samą liczbę różnych wartości predykcyjnych. Maksymalna wartość funkcji w każdym pojemniku jest używana jako punkt przerwania [3, 14].

Ważnym parametrem w procesie wzrostu drzewa CHAID jest wartość p. Wartość p jest miarą używaną do decydowania o tym, które kategorie wartości predykcyjnych mają się łączyć podczas łączenia, a także do decydowania o najlepszym atrybucie podczas dzielenia. Wartość p jest obliczana przy użyciu różnych metod testowania hipotez w zależności od rodzaju zmiennej zależnej (nominalnej, porządkowej lub ciągłej) [3].

6.3 Algorytm ID3

ID3 jest prostym algorytmem uczenia drzewa decyzyjnego opracowanym przez Quinlana (1986). ID3 ma zastosowanie tylko w przypadkach, w których atrybuty (lub cechy) definiujące przykłady danych mają charakter kategoryczny, a przykłady danych należą do wcześniej zdefiniowanych, wyraźnie odróżnialnych (tj. dobrze zdefiniowanych) klas. ID3 to iteracyjny chciwy algorytm, który rozpoczyna się od węzła głównego i ostatecznie buduje całe drzewo. W każdym węźle wybiera się "najlepszy" atrybut do klasyfikacji danych. Atrybut "najlepszy" jest wybierany przy użyciu metryki Information gain. Po wybraniu atrybutu w węźle, przykłady danych w węźle są podzielone na podgrupy na podstawie wartości atrybutów,

które mają. Zasadniczo wszystkie przykłady danych o tej samej wartości atrybutu są umieszczane w tej samej podgrupie. Podgrupy tworzą dzieci obecnego węzła, a algorytm jest powtarzany dla każdego z nowoutworzonych węzłów potomnych. Trwa to dopóki wszystkie elementy danych węzła nie należą do tej samej klasy lub wszystkie atrybuty zostaną wyczerpane [3, 4].

6.4 Algorytm C4.5

Algorytm C4.5 jest rozszerzeniem algorytmu ID3. Ma dodatkową możliwość obsługi ciągłych atrybutów i atrybutów z brakującymi wartościami. Proces budowy drzew w przypadku C4.5 jest taki sam jak w przypadku ID3 - znajdowanie najlepszego podziału w każdym węźle przy użyciu metryki Information gain. Jednak w przypadku atrybutu ciągłego algorytm C4.5 musi wykonać dodatkowy krok przekształcania go w dwuwartościowy atrybut kategoryczny, dzieląc około odpowiedniego progu. Próg ten jest wybierany w taki sposób, że wynikowy podział daje maksymalne wzmocnienie informacji [3, 15].

6.5 Algorytm C5.0

C5.0 to najnowsza wersja Quinlana na podstawie licencji firmowej. Wykorzystuje mniej pamięci i buduje mniejsze zestawy reguł niż C4.5, a jednocześnie jest bardziej dokładna [16].

7 Przycinanie drzew decyzyjnych

Wydajność drzewa można dodatkowo zwiększyć przez przycinanie. Polega ono na usunięciu gałęzi, które korzystają z funkcji o niskim znaczeniu. W ten sposób zmniejszamy złożoność drzewa, a tym samym zwiększamy jego moc predykcyjną, zmniejszając przeuczenie. Przycinanie może rozpocząć się od korzenia lub liści. Najprostsza metoda przycinania rozpoczyna się na liściach i usuwa każdy węzeł z najbardziej popularną klasą w tym liściu, zmiana ta jest zachowana, jeśli nie pogarsza dokładności. Bardziej zaawansowaną metodą przycinania, jest przycinanie kosztów, gdy parametr uczenia (alfa) jest używany do ważenia, czy węzły mogą być usunięte w oparciu o rozmiar pod-drzewa. [6, 10].

8 Biblioteka scikit-learn

8.1 Wstęp

Scikit-learn to bezpłatna biblioteka do uczenia maszynowego z wykorzytaniem języka. Zawiera ona różne algorytmy klasyfikacji, regresji i grupowania, w tym metodę drzew decyzyjnych. Co więcej, bibliteka ta jest przystosowana do współdziałania z numerycznymi i naukowymi bibliotekami NumPy i SciPy w Pythonie. Popularne grupy modeli dostarczane przez scikit-learn to:

- Clustering: do grupowania nieoznakowanych danych, takich jak k-means.
- Cross Validation: do oceny wydajności nadzorowanych modeli na niewidocznych danych.
- Datasets: dla zestawów danych testowych i do generowania zestawów danych o określonych właściwościach do badania zachowania modelu.
- Dimensionality Reduction: w celu zmniejszenia liczby atrybutów w danych do podsumowania, wizualizacji i wyboru funkcji, takich jak analiza głównych składników.
- Ensemble methods: do łączenia przewidywań wielu nadzorowanych modeli.
- Feature extraction: do definiowania atrybutów w obrazie i danych tekstowych.
- Feature selection: do identyfikacji znaczących atrybutów, z których można tworzyć nadzorowane modele.
- Parameter Tuning: aby uzyskać jak najwięcej z nadzorowanych modeli.
- Manifold Learning: do podsumowywania i przedstawiania złożonych wielowymiarowych danych.
- Supervised Models: modele liniowe, analiza dyskryminacyjna, sieci neuronowe, drzewa decyzyjne i wiele więcej.

8.2 Moduł drzew decyzyjnych

Moduł drzew decyzyjnych - sklearn.tree zawiera oparte na drzewach decyzyjnych modele klasyfikacji i regresji. Scikit-learn używa zoptymalizowanej wersji algorytmu CART (Podrozdział 6.1). Klasy zawarte w tym module to:

- tree.**DecisionTreeClassifier**([criterion, ...]) - klasyfikator drzewa decyzyjnego,
- tree.**DecisionTreeRegressor**([criterion, ...]) - regresyjne drzewo decyzyjne,
- tree.**ExtraTreeClassifier**([criterion, ...]) - niezwykle losowy klasyfikator drzewa decyzyjnego,
- tree.**ExtraTreeRegressor**([criterion, ...]) - niezwykle losowe drzewo regresyjne,
- tree.**export_graphviz**(decision_tree , [...]) - eksportowanie drzewa decyzyjngo w formacie DOT.

Drzewa z klas ExtraTreeClassifier i ExtraTreeRegressor różnią się od klasycznych drzew decyzyjnych sposobem ich budowania. Szukając najlepszego podziału, aby podzielić próbki węzła na dwie grupy, losowe podziały są dokonywane dla każdej z możliwych wartości parametru max_features i losowo wybranych cech. Wybierany jest najlepszy podział między tymi elementami. Gdy max_features ma wartość 1, oznacza to utworzenie całkowicie losowego drzewa decyzyjnego.

Najważniejsze z powyższych klas opisano w podrozdziałach 8.2.1 oraz 8.2.2.

8.2.1 Klasa DecisionTreeClassifier

*class sklearn.tree.**DecisionTreeClassifier**(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)*

DecisionTreeClassifier() to funkcja klasyfikująca dla DecisionTree. Stanowi ona konstruktor klasy DecisionTreeClassifier. Ważne parametry to:

- **criterion:** definiuje funkcję pomiaru jakości podziału. Sklearn obsługuje kryteria "gini" dla indeksu Gini i "entropy" dla Information Gain. Domyślnie przyjmuje wartość "gini".
- **splitter:** definiuje strategię wyboru podziału w każdym węźle. Wartość "best" oznacza, że należy wybrać najlepszy podział, a "random", aby wybrać najlepszy podział losowy. Domyślnie przyjmuje wartość "best".
- **max_features:** definiuje ilość funkcji do rozważenia przy poszukiwaniu najlepszego podziału. Można wprowadzić liczbę całkowitą, zmiennoprzecinkową, łańcuchową i wartość None.
  - Jeśli wprowadzona jest liczba całkowita jest ona brana za maksymalną wartość dla każdego podziału.
  - Jeśli zostanie użyta wartość zmiennoprzecinkowa, pokazuje ona procent elementów w każdym podziale.
  - Jeśli zostanie wybrane "auto" lub "sqrt", wówczas max_features = $\sqrt{n\_features}$.
  - Jeśli zostanie wybrany log2, wówczas max_features = $\log_2(n\_features)$.
  - Jeśli nie wprowadzono własnej wartości (wartość domyślna None), wówczas max_features = n_features.
- **max_depth:** parametr określa maksymalną głębokość drzewa. Może przyjmować dowolną liczbę całkowitą lub None. Jeśli ustawiono None, to węzły są rozszerzane, aż wszystkie próbki zostaną wykorzystane lub dopóki wszystkie liście nie zawierają próbek mniejszych niż min_samples_split. Domyślnie przyjmuje wartość None.
- **min_samples_split:** parametr określa minimalną liczbę próbek do podziału. Jeśli zostanie podana wartość całkowita, należy rozważyć min_samples_split jako wartość mininalną, a jeśli podano liczbę zmiennoprzecinkowoą, wówczas pokazuje ona procent próbek. Domyślnie przyjmuje wartość 2.

- **min_samples_leaf:** minimalna liczba próbek, które muszą znajdować się w węźle liści. Jeśli zostanie podana wartość całkowita, należy rozważyć min_samples_leaf jako wartość minimalną, a jeśli zmiennoprzecinkowa, to pokazuje ona procent próbek. Domyślnie przyjmuje wartość 1.
- **max_leaf_nodes:** określa maksymalną liczbę możliwych liści. Jeśli None, pobiera nieograniczoną liczbę węzłów liści. Domyślnie przyjmuje wartość None - bez ograniczeń.
- **min_impurity_split:** definiuje próg wczesnego zatrzymywania wzrostu drzewa. Węzeł zostanie podzielony, jeśli jego nierównomierność przekroczy próg, w przeciwnym razie jest to liść.

Metody klasy DecisionTreeClassifier to:

- **apply(X [, check_input])** - zwraca indeks liścia, dla którego każda próbka jest przewidywana.
- **decision_path(X [, check_input])** - zwraca ścieżkę decyzyjną w drzewie
- **fit(X, y [, sample_weight, check_input, ...])** - zbuduj klasyfikator drzewa decyzyjnego z zestawu treningowego (X, y).
- **get_params([deep])** - pobierz parametry tego estymatora.
- **predict(X [, check_input])** - wyznacz klasę lub wartość regresji dla X.
- **predict_log_proba(X)** - przewiduj log-probabilitie klasy o X próbkach wejściowych.
- **predict_proba(X [, check_input])** - przewiduj prawdopodobieństwo klasowe próbek wejściowych X.
- **score(X, y [, sample_weight])** - zwraca średnią dokładność podanych danych testowych i etykiet.
- **set_params(** params)** - ustaw parametry tego estymatora.

8.2.2 Klasa DecisionTreeRegressor

*class sklearn.tree.**DecisionTreeRegressor**(criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False)*

DecisionTreeRegressor() to funkcja regresyjna dla DecisionTree. Stanowi ona konstruktor klasy DecisionTreeRegressor. Ważne parametry to:

- **criterion:** funkcja pomiaru jakości podziału. Wspierane kryteria to "mse" dla błędu średniej kwadratowej, który jest równy zmniejszeniu wariancji jako kryterium wyboru cech i minimalizuje utratę L2 przy użyciu średniej każdego węzła końcowego, "friedman_mse", który wykorzystuje błąd średniej kwadratowej z wynikiem poprawy Friedmana dla potencjału dzieli się i "mae" na średni błąd bezwzględny, który minimalizuje utratę L1 przy użyciu mediany każdego węzła końcowego.
- **splitter:** strategia używana do wyboru podziału w każdym węźle. Obsługiwane strategie są "best", aby wybrać najlepszy podział i "random", aby wybrać najlepszy podział losowy.
- **max_depth:** maksymalna głębokość drzewa. Jeśli ustawiono None, to węzły są rozszerzane, aż wszystkie próbki zostaną wykorzystane lub dopóki wszystkie liście nie zawierają próbek mniejszych niż min_samples_split.
- **min_samples_split:** minimalna liczba próbek potrzebnych do podzielenia węzła wewnętrznego:
  - Jeśli int, należy rozważyć min_samples_split jako minimalną liczbę.
  - Jeśli zmienna, min_samples_split jest wartością procentową, a suf (min_samples_split * n_samples) to minimalna liczba próbek dla każdego podziału.

- **min_samples_leaf:** minimalna liczba próbek, które muszą znajdować się w węźle liści:
  - Jeśli int, to jako minimum należy przyjąć min_samples_leaf.
  - Jeśli wartość zmiennoprzecinkowa, min_samples_leaf jest wartością procentową, a ceil(min_samples_leaf * n_samples) to minimalna liczba próbek dla każdego węzła.
- **min_weight_fraction_leaf:** minimalna ważona część sumy wag (wszystkich próbek wejściowych), która musi znajdować się w węźle liści. Próbki mają jednakową wagę, gdy nie podano próbki.
- **max_features:** liczba funkcji do rozważenia przy poszukiwaniu najlepszego podziału:
  - Jeśli int, to weź pod uwagę cechy max_features dla każdego podziału.
  - W przypadku wartości zmiennoprzecinkowej, max_features jest wartością procentową, a cechy int (max_features * n_features) są uwzględniane przy każdym podzieleniu.
  - Jeśli "auto", to max_features = n_features.
  - Jeśli "sqrt", to max_features = $\sqrt{n\_features}$.
  - Jeśli "log2", to max_features = $\log_2(n\_features)$.
  - Jeśli brak, to max_features = n_features.
- **random_state:** jeśli int, random_state jest nasieniem używanym przez generator liczb losowych; Jeśli wystąpienie RandomState, random_state jest generatorem liczb losowych; Jeśli nie, generator liczb losowych jest instancją RandomState używaną przez np. random.
- **max_leaf_nodes:** określa maksymalną liczbę możliwych liści. Jeśli None, pobiera nieograniczoną liczbę węzłów liści. Domyślnie przyjmuje wartość None - bez ograniczeń.
- **min_impurity_decrease:** węzeł zostanie podzielony, jeśli podział ten spowoduje zmniejszenie nierównomierności większego lub równego tej wartości.
- **presort:** czy presortować dane, aby przyspieszyć znalezienie najlepszych podziałów w dopasowaniu. W przypadku domyślnych ustawień drzewa decyzyjnego na dużych zestawach danych ustawienie tego na wartość true może spowolnić proces szkolenia. W przypadku korzystania z mniejszego zestawu danych lub ograniczonej głębokości może to przyspieszyć szkolenie.

Metody klasy DecisionTreeRegressor to:

- **apply(X [, check_input])** - zwraca indeks liścia, dla którego każda próbka jest przewidywana.
- **decision_path(X [, check_input])** - zwraca ścieżkę decyzyjną w drzewie
- **fit(X, y [, sample_weight, check_input, ...])** - zbuduj klasyfikator drzewa decyzyjnego z zestawu treningowego (X, y).
- **get_params([deep])** - pobierz parametry tego estymatora.
- **predict(X [, check_input])** - wyznacz klasę lub wartość regresji dla X.
- **score(X, y [, sample_weight])** - zwraca średnią dokładność podanych danych testowych i etykiet.
- **set_params(** params)** - ustaw parametry tego estymatora.

9 Opis wykorzystanych zbiorów danych

W celu przetestowania działania metody drzew decyzyjnych w bibliotece scikit-learn wykorzystano dwa zbiory danych: Forest Fires [17] oraz Mushroom [18].

9.1 Zbiór danych Forest Fires

Zbiór danych zawiera opisy spalonych obszarów lasów, w północno-wschodniej części Portugalii. Zbiór posiada 12 atrybutów kategorycznych lub numerycznych oraz 517 obiektów (próbek) [17]. Definicje wartości atrybutów są następujące:

1. **X:** współrzędna przestrzenna osi "x" na mapie parku Montesinho: od 1 do 9
2. **Y:** współrzędna przestrzenna osi "y" na mapie parku Montesinho: 2 do 9
3. **month:** miesiąc w roku: "jan" do "dec" (od stycznia do grudnia)
4. **day:** dzień tygodnia: "mon" do "sun" (od poniedziałku do niedzieli)
5. **FFMC:** Indeks FFMC (The Fine Fuel Moisture Code) z systemu FWI (Fire Weather Index): 18,7 do 96,20
6. **DMC:** Indeks DMC (Duff Moisture Code) z systemu FWI: 1,1 do 291,3
7. **DC:** Indeks DC (Drought Code) z systemu FWI: 7,9 do 860,6
8. **ISI:** Indeks ISI (Initial Spread Index) z systemu FWI: 0,0 do 56,10
9. **temp**: temperatura w stopniach Celsjusza: od 2,2 do 33,30
10. **RH:** wilgotność względna powietrza w %: 15,0 do 100
11. **wind:** prędkość wiatru w km/h: 0,40 do 9,40
12. **rain:** deszcz w mm/m^2: 0,0 do 6,4
13. **area:** spalony obszar lasu w hektarach: 0,00 do 1090,84.

9.2 Zbiór danych Mushroom

Pobrany zbiór danych Mushroom zapisano w formacie CSV wraz z nazwami nagłówków (atrybutów), zgodnie ze specyfikacją . Zbiór danych zawiera opisy hipotetycznych próbek odpowiadających 23 gatunkom grzybów z rodziny Agaricus i Lepiota. Każdy gatunek jest określony jako jadalny lub trujący. Zbiór posiada 22 kategoryczne atrybuty warunkowe, 1 atrybut decyzyjny oraz 8124 obiekty (próbki) [18]. Definicje wartości atrybutów są następujące:

1. **cap-shape (kształt kapelusza):** bell(dzwon)=b, conical(stożek)=c, convex(wypukły)=x,flat(płaski)=f, knobbed(wybrzuszony)=k, sunken(wklęsły)=s
2. **cap-surface (powierzchnia kapelusza):** fibrous(włóknista)=f, grooves(bruzdowata)=g, scaly(łuskowata)=y, smooth(gładka)=s
3. **cap-color (kolor kapelusza):** brown(brązowy)=n, buff(płowy)=b, cinnamon(cynamonowy)=c, gray(szary)=g, green(zielony)=r, pink(różowy)=p, purple(fioletowy)=u, red(czerwony)=e,white(biały)=w, yellow(żółty)=y
4. **bruises (sińce):** występują=t, nie występują=f
5. **odor (zapach):** almond(migdałowy)=a, anise(anyżowy)=l, creosote(kreozot)=c, fishy(rybi)=y, foul(cuchnący)=f, musty(stęchły)=m, non(brak)=n, pungent(gorzki)=p, spicy(ostry)=s
6. **gill-attachment (przywiązanie do blaszek):** attached(przywiązane)=a, descending(opadające)=d, free(swobodne)=f, notched(wycięte)=n
7. **gill-spacing (odstępy blaszek):** close(bliskie)=c, crowded(zatłoczone)=w, distant(odległe)=d
8. **gill-size (wielkość blaszek):** broad(szerokie)=b, narrow(wąskie)=n
9. **gill-color (kolor blaszek):** black(czarne)=k, brown(brązowe)=n, buff(płowe)=b, chocolate(czekoladowe)=h, gray(szare)=g, green(zielone)=r, orange(pomarańczowe)=o, pink(różowe)=p, purple(fioletowe)=u, red(czerwone)=e, white(białe)=w, yellow(żółte)=y

10. **stalk-shape (kształt łodygi):** enlarging(rozszerzający się)=e, tapering(spiczasty)=t
11. **stalk-root (korzeń łodygi)*:** bulbous(bulwiasty)=b, club(maczuga)=c, cup(kielich)=u, equal(jednakowy)=e, rhizomorphs(ryzomorfy)=z, rooted(ukorzeniony)=r, missing(brak)=?
12. **stalk-surface-above-ring (powierzchnia łodygi powyżej pierścienia):** fibrous(włóknista)=f, scaly(łuskowata)=y, silky(jedwabista)=k, smooth(gładka)=s
13. **stalk-surface-below-ring (powierzchnia łodygi poniżej pierścienia):** fibrous(włóknista)=f, scaly(łuskowata)=y, silky(jedwabista)=k, smooth(gładka)=s
14. **stalk-color-above-ring (kolor łodygi powyżej pierścienia):** brown(brązowy)=n,buff(płowy)=b,cinnamon(cynamonowy)=c, gray(szary)=g, orange(pomarańczowy)=o, pink(różowy)=p, red(czerwony)=e, white(biały)=w, yellow(żółty)=y
15. **stalk-color-below-ring (kolor łodygi pod pierścieniem):** brown(brązowy)=n, buff(płowy)=b,cinnamon(cynamonowy)=c, gray(szary)=g, orange(pomarańczowy)=o, pink(różowy)=p, red(czerwony)=e, white(biały)=w, yellow(żółty)=y
16. **veil-type (rodzaj zasłony):** partial(częściowa)=p, universal(powszechna)=u
17. **veil-color (kolor zasłony):** brown(brązowy)=n, orange(pomarańczowy)=o, white(biały)=w, yellow(żółty)=y
18. **ring-number (ilość pierścieni):** none(brak)=n, one(jeden)=o, two(dwa)=t
19. **ring-type (typ pierścienia):** cobwebby(pajęczynowy)=c, evanescent(zanikający)=e, flaring(kloszowy)=f, large(duży)=l, none(brak)=n, pendant(wiszący)=p, sheathing(poszycie)=s,zone(pas)=z
20. **spore-print-color (kolor wysypu zarodników):** black(czarny)=k, brown(brązowy)=n, buff(płowy)=b, chocolate(czekoladowy)=h, green(zielony)=r, orange(pomarańczowy)=o, purple(fioletowy)=u, white(biały)=w, yellow(żółty)=y
21. **population (populacja):** abundant(obfita)=a, clustered(skupiona)=c, numerous(liczna)=n, scattered(rzadka)=s, several(kilka)=v, solitary(odludna)=y
22. **habitat (środowisko):** grasses(trawy)=g, leaves(liście)=l, meadows(łąki)=m, paths(dróżki)=p, urban(miejskie)=u, waste(pustynne)=w, woods(las)=d

* - atrybut zawiera 2480 brakujących wartości, oznaczonych znakiem zapytania.

Podział klasy decyzyjnej jest następujący:

- grzyby jadalne: 4208 (51,8%),
- grzyby trujące: 3916 (48,2%),
- **łącznie: 8124 próbek**.

10 Opis programów i wyników

Obydwa programy stworzono przy użyciu języka Python, bibliteki scikit-learn i kilku innych oraz środowiska Jupyter notebook, uruchamianego za pomocą polecenia 'jupyter notebook' w konsoli.

10.1 Zbiór danych Forest Fires

Załadowanie bibliotek

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        %matplotlib inline
        from matplotlib import pyplot as plt

        #Ustalenie stylu wykresów jako ggplot
        #plt.style.use('ggplot')
        from sklearn import preprocessing
        from sklearn import tree
        import sklearn.metrics as metrics
        from sklearn.model_selection import train_test_split
        import pydotplus
        from IPython.display import Image
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import r2_score, mean_absolute_error
```

Wczytanie danych

```
In [2]: def dataframe_size_formated(dataframe, extra=""):
            print("Rozmiar danych{}: ".format(extra), dataframe.shape)

        # Ustalenie ścieżki do datasetu
        filename_forestfires = './forestfires.csv'

        # Wczytanie datasetu jako dataframe
        forestfires_dataframe = pd.read_csv(filename_forestfires, sep=";")

        # Wyświetlenie dataframe
        display(forestfires_dataframe)
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00

3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00
5	8	6	aug	sun	92.3	85.3	488.0	14.7	22.2	29	5.4	0.0	0.00
6	8	6	aug	mon	92.3	88.9	495.6	8.5	24.1	27	3.1	0.0	0.00
7	8	6	aug	mon	91.5	145.4	608.2	10.7	8.0	86	2.2	0.0	0.00
8	8	6	sep	tue	91.0	129.5	692.6	7.0	13.1	63	5.4	0.0	0.00
9	7	5	sep	sat	92.5	88.0	698.6	7.1	22.8	40	4.0	0.0	0.00
10	7	5	sep	sat	92.5	88.0	698.6	7.1	17.8	51	7.2	0.0	0.00
11	7	5	sep	sat	92.8	73.2	713.0	22.6	19.3	38	4.0	0.0	0.00
12	6	5	aug	fri	63.5	70.8	665.3	0.8	17.0	72	6.7	0.0	0.00
13	6	5	sep	mon	90.9	126.5	686.5	7.0	21.3	42	2.2	0.0	0.00
14	6	5	sep	wed	92.9	133.3	699.6	9.2	26.4	21	4.5	0.0	0.00
15	6	5	sep	fri	93.3	141.2	713.9	13.9	22.9	44	5.4	0.0	0.00
16	5	5	mar	sat	91.7	35.8	80.8	7.8	15.1	27	5.4	0.0	0.00
17	8	5	oct	mon	84.9	32.8	664.2	3.0	16.7	47	4.9	0.0	0.00
18	6	4	mar	wed	89.2	27.9	70.8	6.3	15.9	35	4.0	0.0	0.00
19	6	4	apr	sat	86.3	27.4	97.1	5.1	9.3	44	4.5	0.0	0.00
20	6	4	sep	tue	91.0	129.5	692.6	7.0	18.3	40	2.7	0.0	0.00
21	5	4	sep	mon	91.8	78.5	724.3	9.2	19.1	38	2.7	0.0	0.00
22	7	4	jun	sun	94.3	96.3	200.0	56.1	21.0	44	4.5	0.0	0.00
23	7	4	aug	sat	90.2	110.9	537.4	6.2	19.5	43	5.8	0.0	0.00
24	7	4	aug	sat	93.5	139.4	594.2	20.3	23.7	32	5.8	0.0	0.00
25	7	4	aug	sun	91.4	142.4	601.4	10.6	16.3	60	5.4	0.0	0.00
26	7	4	sep	fri	92.4	117.9	668.0	12.2	19.0	34	5.8	0.0	0.00
27	7	4	sep	mon	90.9	126.5	686.5	7.0	19.4	48	1.3	0.0	0.00
28	6	3	sep	sat	93.4	145.4	721.4	8.1	30.2	24	2.7	0.0	0.00
29	6	3	sep	sun	93.5	149.3	728.6	8.1	22.8	39	3.6	0.0	0.00
..	..	..	...	...	...	...	...	...	...	..	...	...	...
487	5	4	aug	tue	95.1	141.3	605.8	17.7	26.4	34	3.6	0.0	16.40
488	4	4	aug	tue	95.1	141.3	605.8	17.7	19.4	71	7.6	0.0	46.70
489	4	4	aug	wed	95.1	141.3	605.8	17.7	20.6	58	1.3	0.0	0.00
490	4	4	aug	wed	95.1	141.3	605.8	17.7	28.7	33	4.0	0.0	0.00
491	4	4	aug	thu	95.8	152.0	624.1	13.8	32.4	21	4.5	0.0	0.00
492	1	3	aug	fri	95.9	158.0	633.6	11.3	32.4	27	2.2	0.0	0.00
493	1	3	aug	fri	95.9	158.0	633.6	11.3	27.5	29	4.5	0.0	43.32
494	6	6	aug	sat	96.0	164.0	643.0	14.0	30.8	30	4.9	0.0	8.59
495	6	6	aug	mon	96.2	175.5	661.8	16.8	23.9	42	2.2	0.0	0.00

```
496  4  5   aug  mon  96.2  175.5  661.8  16.8  32.6  26   3.1   0.0   2.77
497  3  4   aug  tue  96.1  181.1  671.2  14.3  32.3  27   2.2   0.0  14.68
498  6  5   aug  tue  96.1  181.1  671.2  14.3  33.3  26   2.7   0.0  40.54
499  7  5   aug  tue  96.1  181.1  671.2  14.3  27.3  63   4.9   6.4  10.82
500  8  6   aug  tue  96.1  181.1  671.2  14.3  21.6  65   4.9   0.8   0.00
501  7  5   aug  tue  96.1  181.1  671.2  14.3  21.6  65   4.9   0.8   0.00
502  4  4   aug  tue  96.1  181.1  671.2  14.3  20.7  69   4.9   0.4   0.00
503  2  4   aug  wed  94.5  139.4  689.1  20.0  29.2  30   4.9   0.0   1.95
504  4  3   aug  wed  94.5  139.4  689.1  20.0  28.9  29   4.9   0.0  49.59
505  1  2   aug  thu  91.0  163.2  744.4  10.1  26.7  35   1.8   0.0   5.80
506  1  2   aug  fri  91.0  166.9  752.6   7.1  18.5  73   8.5   0.0   0.00
507  2  4   aug  fri  91.0  166.9  752.6   7.1  25.9  41   3.6   0.0   0.00
508  1  2   aug  fri  91.0  166.9  752.6   7.1  25.9  41   3.6   0.0   0.00
509  5  4   aug  fri  91.0  166.9  752.6   7.1  21.1  71   7.6   1.4   2.17
510  6  5   aug  fri  91.0  166.9  752.6   7.1  18.2  62   5.4   0.0   0.43
511  8  6   aug  sun  81.6   56.7  665.6   1.9  27.8  35   2.7   0.0   0.00
512  4  3   aug  sun  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0   6.44
513  2  4   aug  sun  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  54.29
514  7  4   aug  sun  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  11.16
515  1  4   aug  sat  94.4  146.0  614.7  11.3  25.6  42   4.0   0.0   0.00
516  6  3   nov  tue  79.5    3.0  106.7   1.1  11.8  31   4.5   0.0   0.00

[517 rows x 13 columns]
```

```
In [3]: dataframe_size_formated(forestfires_dataframe)

Rozmiar danych:  (517, 13)
```

Zbiór danych ma 517 wierszy i 13 kolumn (ostatnia kolumna to atrybut decyzyjny, a pozostałe 12 kolumn to atrybuty warunkowe). W celu dalszego zbadania datasetu i weryfikacji typów danych kategorycznych w każdej kolumnie, wypisano unikalne wartości każdej kolumny. Sprawdzono również, czy zbiór danych zawiera brakujące wartości lub niepotrzebne kolumny.

```
In [4]: def attributes_count(dataframe):
            print("Liczba różnych wartości atrybutów dla każdej kolumny:")
            for x in dataframe.columns:
                uniq = dataframe[x].unique()
                print("{:>8}: {:>2}".format(x, uniq.shape[0]))
```

```
        attributes_count(forestfires_dataframe)
```

```
Liczba różnych wartości atrybutów dla każdej kolumny:
       X:  9
       Y:  7
   month: 12
     day:  7
    FFMC: 106
     DMC: 215
      DC: 219
     ISI: 119
    temp: 192
      RH: 75
    wind: 21
    rain:  7
    area: 251
```

Zauważono, że spośród 12 atrybutów warunkowych, 4 z nich mają liczbę klas mniejeszą niż 10. Również atrybut month nie charakteryzuje się dużą liczbą klas. Z uwagi na chęć wyeliminowania zalezności modelu od położenia, dnia tygodnia i opadów deszczu, podjęto decyzję o usunięciu kolumn o liczbie klas mniejszej równej 10, w tym celu utworzono poniższą funkcję.

Wartości atrybutu month przekształcono w następujący sposób: 'jan'=1, 'feb'=2, . . . , 'dec'=12.

```
In [5]: def del_with_classes_no_less_than(dataframe, less):
            for col in dataframe.columns.values:
                col_unique = dataframe[col].unique()
                if len(col_unique) <= less:
                    print("Usunięto kolumnę '{}',która zawiera liczbę klas mnieszą równą {}: {}".format(col, less, col_unique))
                    dataframe = dataframe.drop(col, 1)
            return dataframe
```

```
In [6]: dataframe_size_formated(forestfires_dataframe, " przed usunięciem atrybutów")
        forestfires_dataframe = del_with_classes_no_less_than(forestfires_dataframe, 10)
        dataframe_size_formated(forestfires_dataframe, " po usunięciu atrybutów")
```

```
        forestfires_dataframe.month = forestfires_dataframe.month.map({
            'jan': 1,
            'feb': 2,
            'mar': 3,
            'apr': 4,
            'may': 5,
            'jun': 6,
            'jul': 7,
            'aug': 8,
            'sep': 9,
            'oct': 10,
            'nov': 11,
            'dec': 12,
        })
```

```
Rozmiar danych przed usunięciem atrybutów:  (517, 13)
Usunięto kolumnę 'X',która zawiera liczbę klas mnieszą równą 10: [7 8 6 5 4 2 9 1 3]
Usunięto kolumnę 'Y',która zawiera liczbę klas mnieszą równą 10: [5 4 6 3 2 9 8]
Usunięto kolumnę 'day',która zawiera liczbę klas mnieszą równą 10: ['fri' 'tue' 'sat' 'sun' 'mon' 'wed' 'thu']
Usunięto kolumnę 'rain',która zawiera liczbę klas mnieszą równą 10: [ 0.   0.2  1.   6.4  0.8  0.4  1.4]
Rozmiar danych po usunięciu atrybutów:  (517, 9)
```

In [7]:
```
attributes_count(forestfires_dataframe)
```

```
Liczba różnych wartości atrybutów dla każdej kolumny:
   month: 12
    FFMC: 106
     DMC: 215
      DC: 219
     ISI: 119
    temp: 192
      RH: 75
    wind: 21
    area: 251
```

Z uwagi na brak danych kategorycznych w oczyszczonym zbiorze, kodowanie wartości atrybutów (kolumn) nie jest konieczne. Dokonać podziału danych na atrybuty warunkowe (zmienna X) i decyzyjne (zmienna Y).

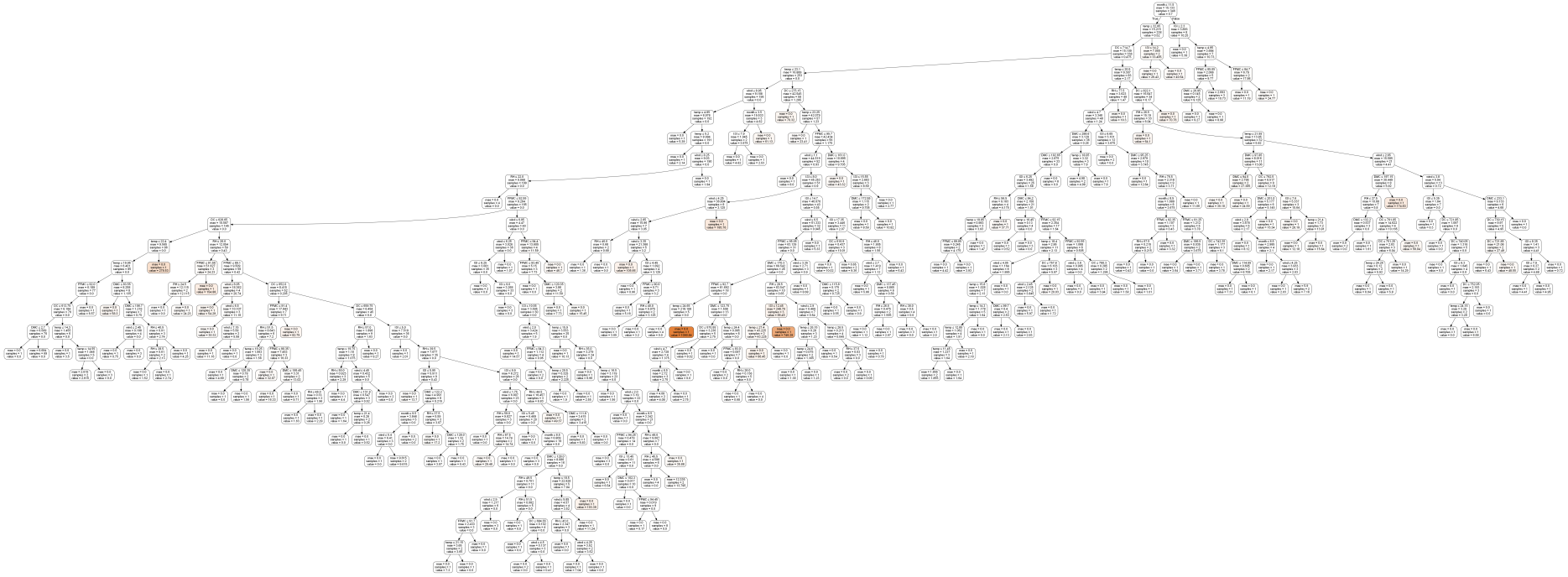
```
In [8]: X = forestfires_dataframe.drop(['area'], axis=1)
        Y = forestfires_dataframe['area']
```

Kolejnym podziałem, który należy wykonać, jest podział danych na część treningową i testową. Założono, że rozmiar części testowej będzie wynosił 33% wszystkich danych. W celu zachowania powtarzalności wyników parametr random_state ustawiono na wartość 34 (ustawienie innej wartości bedzie powodowało wygenerowanie innego podziału danych i innego drzewa decyzyjnego).

```
In [9]: random_state = 34

        X_train, X_test ,Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state=random_state)
```

W oparciu o dane zbudowano drzewo decyzyjne. Do tego celu utworzno specjalną funkcję.

```
In [10]: def build_tree(X, X_train, X_test, Y_train, Y_test, random_state, **kwargs):
             regr = tree.DecisionTreeRegressor(random_state=random_state, **kwargs)
             regr = regr.fit(X_train, Y_train)

             dot_data = tree.export_graphviz(regr, out_file=None,
                                             feature_names=X.columns,
                                             filled=True, rounded=True,
                                             special_characters=True)
             graph = pydotplus.graph_from_dot_data(dot_data)
             display(Image(graph.create_png()))

             return regr

In [11]: regr = build_tree(X, X_train, X_test, Y_train, Y_test, random_state, max_depth=None, criterion='mae')
```



Drzewo wygenerowane bez przycinania oraz innych optymalizacji jest zbyt duże.

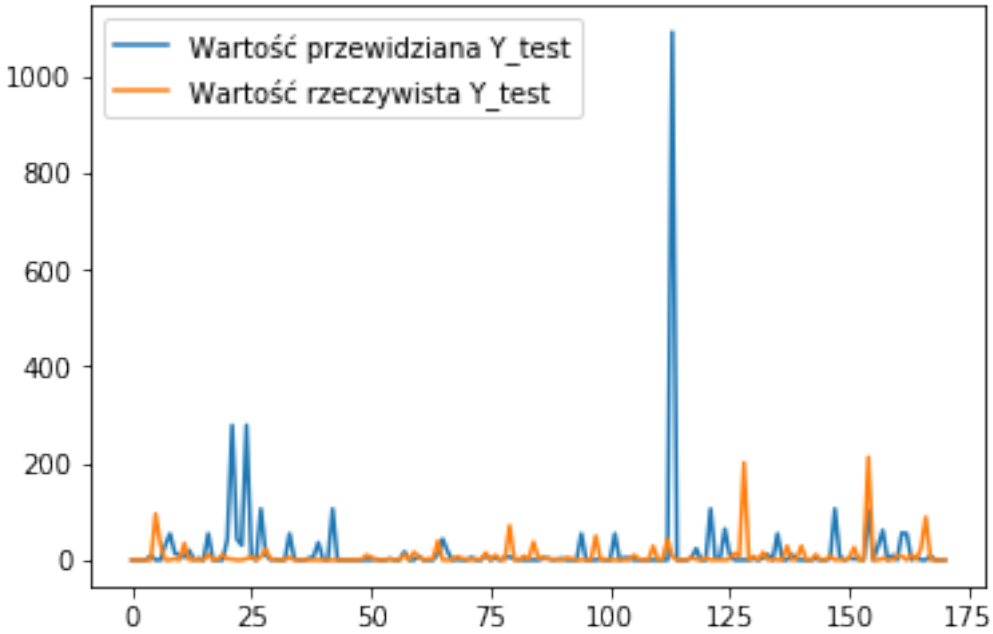
```
In [12]: def check_tree_mae(regr, X_train, X_test, Y_train, Y_test, print_score=True):
             preds=regr.predict(X_test)

             line_pred, = plt.plot(preds, label='Wartość przewidziana Y_test')
             line_real, = plt.plot(Y_test.values, label='Wartość rzeczywista Y_test')
             plt.legend(handles=[line_pred, line_real])

             mae = mean_absolute_error(Y_test, preds)
             if print_score:
                 print('Mean absolute error: %.3f' % mae)
```

```
        return mae

    def check_tree_cross_val_score(regr, X, Y, print_score=True):
        scores = cross_val_score(regr, X, Y, cv=3, scoring='neg_mean_squared_error')
        current_score = np.mean(np.sqrt(-scores))

        if print_score:
            print('Dokładność pomiędzy Y_pred oraz Y_test (neg_mean_squared_error): %.3f' % current_score)
        return current_score
```

Metody mean absolute error (MAE) oraz mean squared error (MSE) zostały wykorzystane do określenia dokładności między Y_pred a Y_test. Im mniejsza wartość obydwu parametrów tym drzewo decyzyjne jest lepiej dopasowane do danych testowych. MAE jest miarą różnicy między dwiema zmiennymi ciągłymi. Z kolei MSE jest wartością oczekiwaną kwadratu „błędu", czyli różnicy pomiędzy estymatorem i wartością estymowaną.

```
In [13]: check_tree_mae(regr, X_train, X_test, Y_train, Y_test)
         check_tree_cross_val_score(regr, X, Y)

         print("")
```

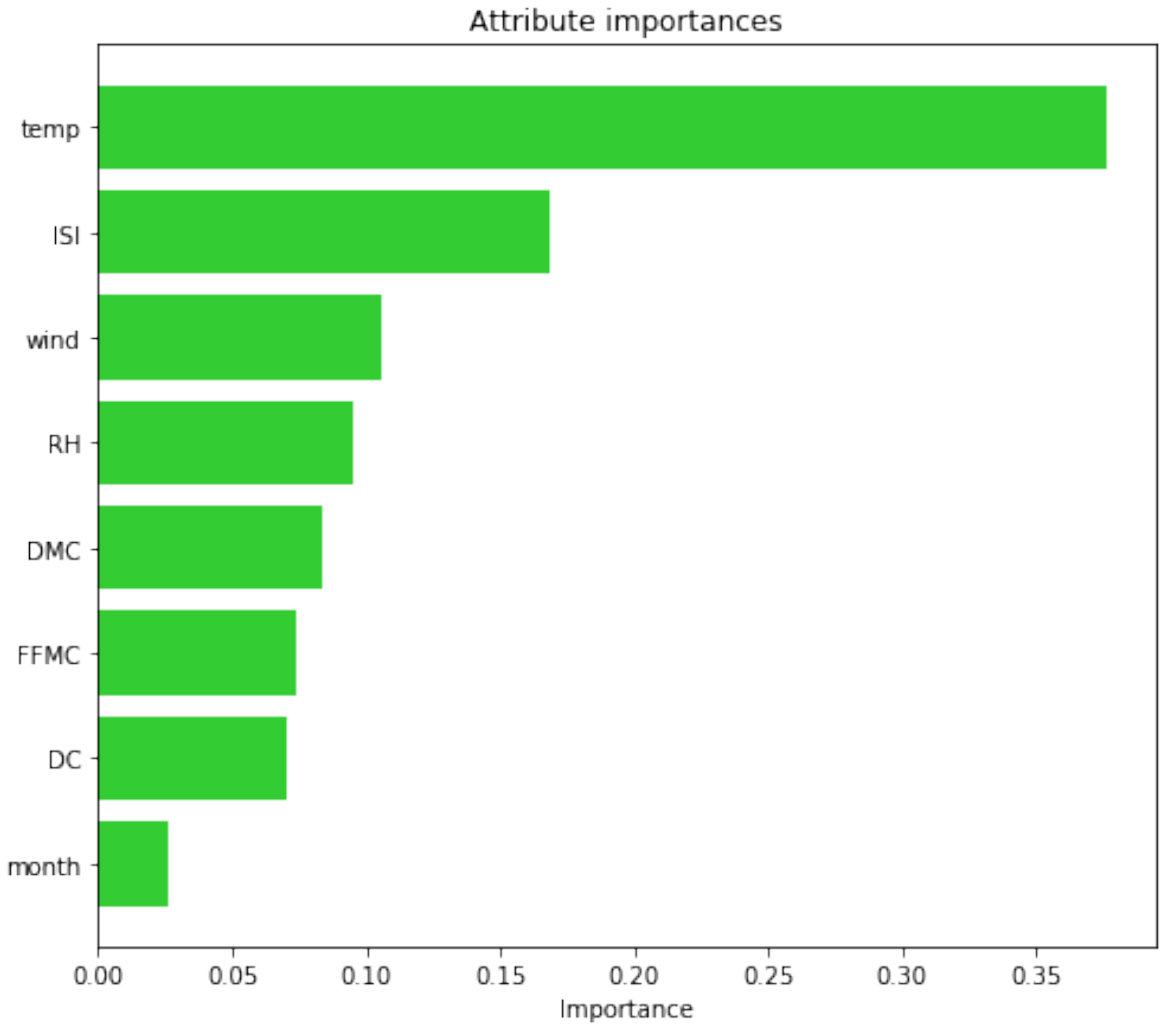
```
Mean absolute error: 24.524
Dokładność pomiędzy Y_pred oraz Y_test (neg_mean_squared_error): 108.680
```



Z wartości Mean absolute error widać, że błąd drzewa z maksymalnie 10 poziomami wynosi średnio około 24,5 hektara lasu na przewidywanie. Również bład MSE jest dość duży.

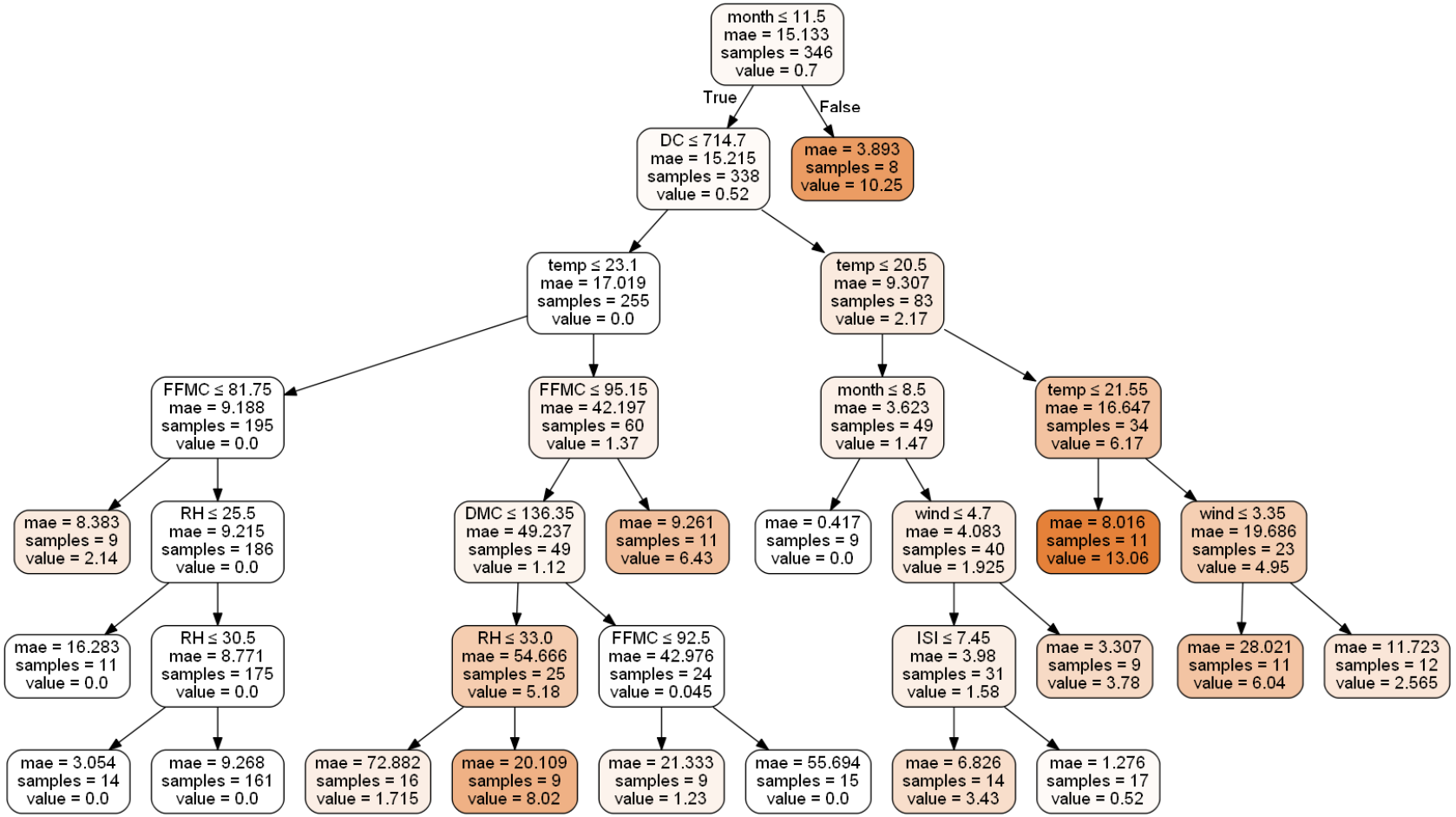
```
In [14]: def attribute_importance(regr, X):
             attrs = X.columns.values
             attr_importance = regr.feature_importances_
             sorted_attr_importance = np.argsort(attr_importance)
             range_sorted_attr_importance = range(len(sorted_attr_importance))

             plt.figure(figsize=(8, 7))
             plt.barh(range_sorted_attr_importance, attr_importance[sorted_attr_importance], color='#33cc33')
             plt.yticks(range_sorted_attr_importance, attrs[sorted_attr_importance])
             plt.xlabel('Importance')
             plt.title('Attribute importances')
             plt.draw()
             plt.show()
```

```
In [15]: attribute_importance(regr, X)
```



Zauważono, że najważnijszymi atrybutami są: month oraz temp. Mniejsze znaczenie mają pozostałe atrybuty. Wygenerowane drzewo decyzyjne posiada głębokość powyżej 15, jest to zdecydowanie zbyt dużo. Optymalne drzewo przedstawiono poniżej.

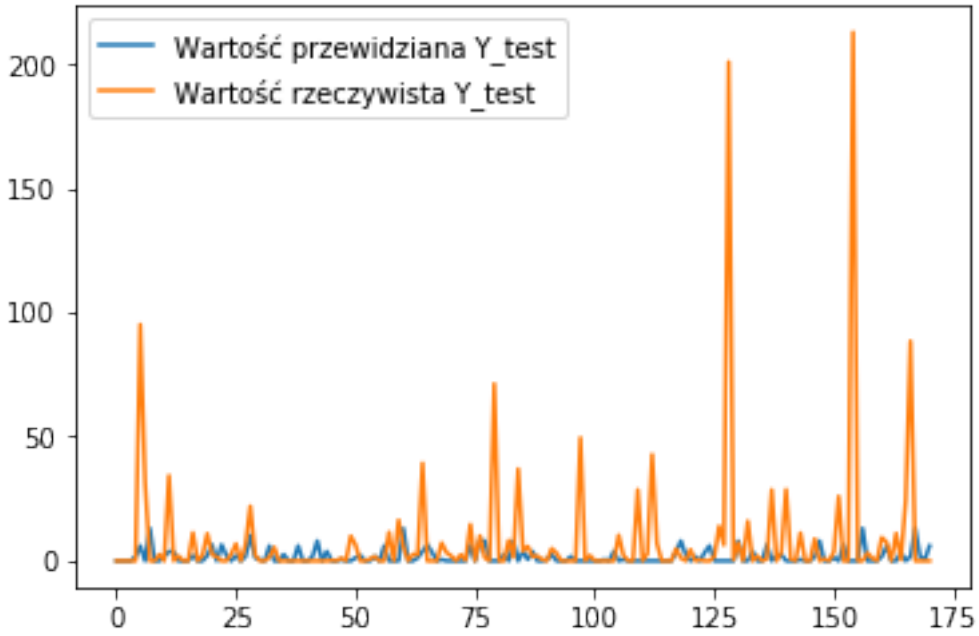
In [16]:
```
regr = build_tree(X, X_train, X_test, Y_train, Y_test, random_state, max_depth=6, min_samples_leaf = 8, criterion='mae')
```



In [17]:
```
check_tree_mae(regr, X_train, X_test, Y_train, Y_test)
check_tree_cross_val_score(regr, X, Y)
```

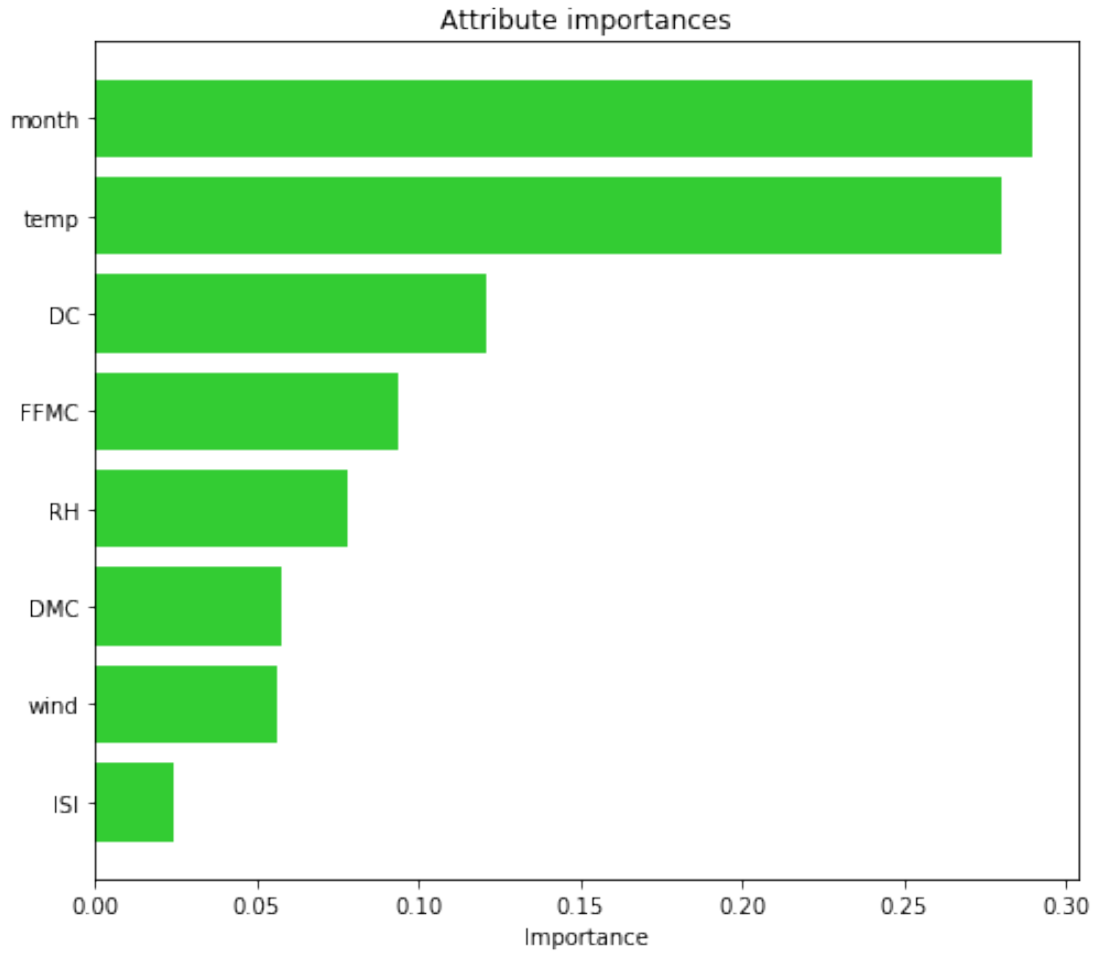
```
        print("")
```

```
Mean absolute error: 8.706
Dokładność pomiędzy Y_pred oraz Y_test (neg_mean_squared_error): 58.401
```



Z wartości Mean absolute error widać, że błąd drzewa z maksymalnie 10 poziomami wynosi średnio około 8,7 hektara lasu na przewidywanie. Stwierdzono, że drzewo jest dość dobrze dopasowane do danych w porównaniu do poprzedniego drzewa.

```
In [18]: attribute_importance(regr, X)
```



Wygenerowane drzewo decyzyjne posiada głębokość wynoszącą 6 (drzewo bez ograniczenia wysokości głębokość powyżej 15). Zauważono, że najważniejszymi najważnijesze atrybuty nie zmieniły się w stosunku do poprzedniego drzewa.

10.2 Zbiór danych Mushroom

Załadowanie bibliotek

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        %matplotlib inline
        from matplotlib import pyplot as plt

        #Ustalenie stylu wykresów jako ggplot
        # plt.style.use('ggplot')
        from sklearn import preprocessing
        from sklearn import tree
        import sklearn.metrics as metrics
        from sklearn.model_selection import train_test_split
        import pydotplus
        from IPython.display import Image
```

Wczytanie danych

```
In [2]: # Ustalenie ścieżki do datasetu
        filename_mushrooms = './agaricus-lepiota.csv'

        # Wczytanie datasetu jako dataframe
        mushrooms_dataframe = pd.read_csv(filename_mushrooms, sep=";")

        # Wyświetlenie dataframe
        display(mushrooms_dataframe)
```

Zbiór danych ma 8124 wiersze i 23 kolumny (pierwsza kolumna to atrybut decyzyjny, a pozostałe 22 kolumny to atrybuty warunkowe). W celu dalszego zbadania datasetu i weryfikacji typów danych kategorycznych w każdej kolumnie, wypisano unikalne wartości każdej kolumny. Sprawdzono również, czy zbiór danych zawiera brakujące wartości lub niepotrzebne kolumny.

```
In [3]: print("Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny:")
        for x in mushrooms_dataframe.columns:
            x_unique = mushrooms_dataframe[x].unique()
            print("{:>25}: {:>2} {}".format(x, x_unique.shape[0], x_unique))
```

```
Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny:
                 classes:  2 ['p' 'e']
               cap-shape:  6 ['x' 'b' 's' 'f' 'k' 'c']
             cap-surface:  4 ['s' 'y' 'f' 'g']
               cap-color: 10 ['n' 'y' 'w' 'g' 'e' 'p' 'b' 'u' 'c' 'r']
```

```
                 bruises:  2 ['t' 'f']
                    odor:  9 ['p' 'a' 'l' 'n' 'f' 'c' 'y' 's' 'm']
          gill-attachment:  2 ['f' 'a']
             gill-spacing:  2 ['c' 'w']
                gill-size:  2 ['n' 'b']
               gill-color: 12 ['k' 'n' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o']
              stalk-shape:  2 ['e' 't']
               stalk-root:  5 ['e' 'c' 'b' 'r' '?']
 stalk-surface-above-ring:  4 ['s' 'f' 'k' 'y']
 stalk-surface-below-ring:  4 ['s' 'f' 'y' 'k']
   stalk-color-above-ring:  9 ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y']
   stalk-color-below-ring:  9 ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c']
                veil-type:  1 ['p']
               veil-color:  4 ['w' 'n' 'o' 'y']
              ring-number:  3 ['o' 't' 'n']
                ring-type:  5 ['p' 'e' 'l' 'f' 'n']
        spore-print-color:  9 ['k' 'n' 'u' 'h' 'w' 'r' 'o' 'y' 'b']
               population:  6 ['s' 'n' 'a' 'v' 'y' 'c']
                  habitat:  7 ['u' 'g' 'm' 'd' 'p' 'w' 'l']
```

Zauważono, że spośród 22 atrybutów warunkowych, jedynie ‘veil-type’ zawiera tylko jedną wartość “p”. Zatem atrybut ten nie zapewnia żadnej wartości dodanej do klasyfikatora. Podjęto decyzję o usunięciu tej kolumny - utworzono generyczny kod usuwający wszystkie kolumny zawierające jedną wartość.

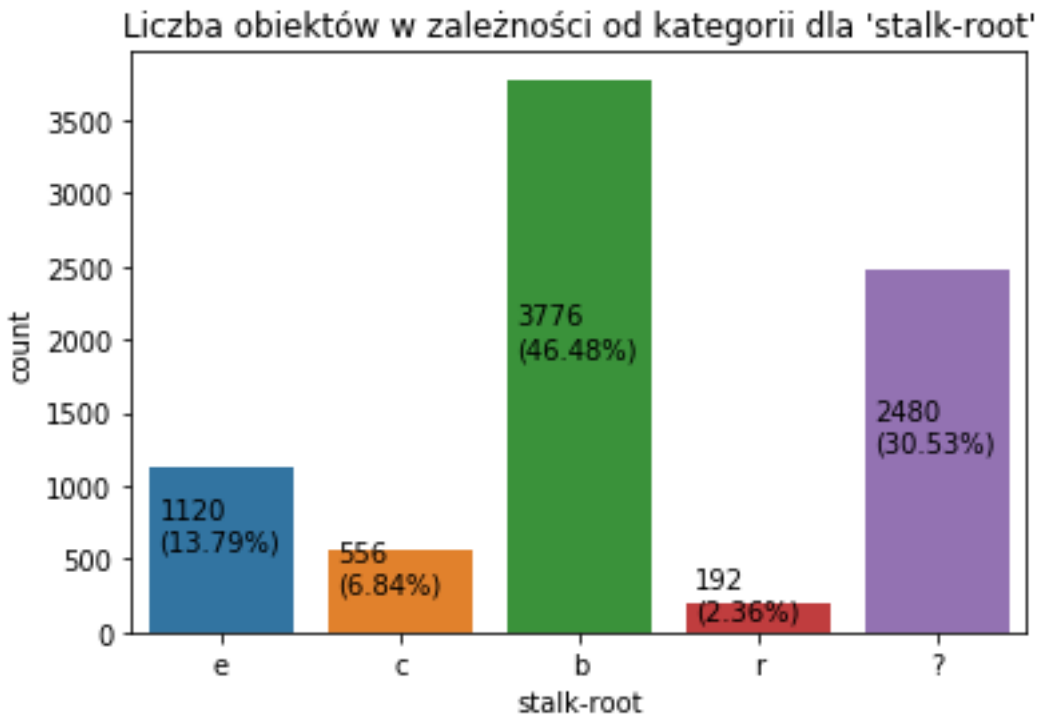
```
In [4]: print("Rozmiar mushrooms_dataframe przed usunięciem: ",mushrooms_dataframe.shape)

        # Usunięcie kolumn zawierających jedną wartośc
        for col in mushrooms_dataframe.columns.values:
            col_unique = mushrooms_dataframe[col].unique()
            if len(col_unique) == 1:
                print("Usunięto kolumnę '{}',która zawiera tylko jedną wartość: {}".format(col, col_unique[0]))
                mushrooms_dataframe = mushrooms_dataframe.drop(col, 1)

        print("Rozmiar mushrooms_dataframe po usunięciu: ",mushrooms_dataframe.shape)
```

```
Rozmiar mushrooms_dataframe przed usunięciem:  (8124, 23)
Usunięto kolumnę 'veil-type',która zawiera tylko jedną wartość: p
Rozmiar mushrooms_dataframe po usunięciu:  (8124, 22)
```

Stwierdzono również, że kolumna ‘stalk-root’ zawiera brakujące wartości. Zbadano udział brakujących wartości w zbiorze - utworzono generyczny kod badający udziały brakujących wartości.

```
In [5]: for x in mushrooms_dataframe.columns:
            x_unique = mushrooms_dataframe[x].unique()
            if '?' in x_unique:
                column = mushrooms_dataframe[x]
                column_count = column.count()
                column_value_count = column.value_counts()

                print("Liczba obiektów w zależności od kategorii i ich udział procentowy dla klasy '{}':\n".format(x))
                stat = column_value_count.to_frame()
                stat['percent'] = 100. * column_value_count / column_count
                print(stat)

                fig = plt.figure()
                fig.patch.set_facecolor('xkcd:white')
                ax = sns.countplot(x=x, data=mushrooms_dataframe)
                ax.set_title("Liczba obiektów w zależności od kategorii dla '{}'".format(x))

        #         for p in ax.patches:
        #             height = p.get_height()
        #             ax.text(p.get_x()+0.25, height+ 3, 'n=%.0f'%(height))

                for p in ax.patches:
                    ax.annotate('{:.0f}\n({:.2f}%)'.format(p.get_height(),
                                                           100. * p.get_height() / column_count),
                                                           (p.get_x()+0.05, p.get_height()//2))
                plt.show()
```

```
Liczba obiektów w zależności od kategorii i ich udział procentowy dla klasy 'stalk-root':

   stalk-root    percent
b        3776  46.479567
?        2480  30.526834
e        1120  13.786312
c         556   6.843919
r         192   2.363368
```



Możliwe działania do podjęcia w pzypadku występowania brakujących danych to m.in. usunięcie kolumn lub wierszy zawierających brakujące dane, wypełninie brakujących wartości inną wartościa np. z poprzedniej lub następnej komórki. Stwierdzono, że udzial procentowy brakujących wartości ('?') dla atrybutu 'stalk-root' wynosi ponad 30,5%. Podjeto decyzję sporządzeniu dwóch wersji zbioru danych: z usuniętymi wierszami oraz z usuniętymi kolumnami zawierającymi brakujące wartości.

Utworzono generyczny kod oczyszczający zbiór danych z wierszy zawierających brakujące wartości:

```
In [6]: # Wykonanie kopii danych
        mushrooms_dataframe_dropped_rows = mushrooms_dataframe.copy(deep=True)

        # Usunięcie wierszy
        for x in mushrooms_dataframe_dropped_rows.columns:
            to_delete_count = mushrooms_dataframe_dropped_rows[mushrooms_dataframe_dropped_rows[x] == '?'].shape[0]
            if to_delete_count > 0:
                mushrooms_dataframe_dropped_rows = mushrooms_dataframe_dropped_rows[mushrooms_dataframe_dropped_rows[x] != '?']
                print("W kolumnie '{}' usunięto {} wierszy zawierających brakujące wartości.".format(x, to_delete_count))
```

```
        print("mushrooms_dataframe_dropped_rows: ",mushrooms_dataframe_dropped_rows.shape)

        print("\n Podział atrybutu decyzyjnego:")
        print(mushrooms_dataframe_dropped_rows['classes'].value_counts())
```

```
W kolumnie 'stalk-root' usunięto 2480 wierszy zawierających brakujące wartości.
mushrooms_dataframe_dropped_rows:  (5644, 22)

 Podział atrybutu decyzyjnego:
e    3488
p    2156
Name: classes, dtype: int64
```

Utworzono generyczny kod oczyszczający zbiór danych z kolumn zawierających brakujące wartości powyżej zadanego progu procentowego (25%):

```
In [7]: # Próg procentowy usuwania kolumn z brakującymi wartościami
        drop_percentage = 0.25

        # Wykonanie kopii danych
        mushrooms_dataframe_dropped_cols = mushrooms_dataframe.copy(deep=True)

        # Zastąpienie znaku ? wartością nan
        for col in mushrooms_dataframe_dropped_cols:
            mushrooms_dataframe_dropped_cols.loc[mushrooms_dataframe_dropped_cols[col] == '?', col] = np.nan

        # Usunięcie kolumn
        for col in mushrooms_dataframe_dropped_cols.columns.values:
            no_rows = mushrooms_dataframe_dropped_cols[col].isnull().sum()
            percentage = no_rows / mushrooms_dataframe_dropped_cols.shape[0]
            if percentage >= drop_percentage:
                del mushrooms_dataframe_dropped_cols[col]
                print("Kolumna '{}' zawierająca {} brakujących wartości ({}% zbioru) została usunięta.".format(col, no_rows, percentage))

        print("mushrooms_dataframe_dropped_cols: ",mushrooms_dataframe_dropped_rows.shape)

        print("\n Podział atrybutu decyzyjnego:")
        print(mushrooms_dataframe_dropped_cols['classes'].value_counts())
```

```
Kolumna 'stalk-root' zawierająca 2480 brakujących wartości (0.3052683407188577% procent zbioru) została usunięta.
mushrooms_dataframe_dropped_cols:  (5644, 22)
```

```
 Podział atrybutu decyzyjnego:
e    4208
p    3916
Name: classes, dtype: int64
```

Przed przystąpieniem do budowy drzewa decyzyjnego należy zakodować wartości atrybutów (kolumn). Do zakodowania wartości kategorycznych użyta zostanie technika kodowania etykiet, która konwertuje każdą wartość w kolumnie na liczbę.

```
In [8]: print('Kodowanie danych z usuniętymi wierszami:')
        le_rows = preprocessing.LabelEncoder()
        for column in mushrooms_dataframe_dropped_rows.columns:
            mushrooms_dataframe_dropped_rows[column] = le_rows.fit_transform(mushrooms_dataframe_dropped_rows[column])

        print("Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny po zakodowaniu:")
        for x in mushrooms_dataframe_dropped_rows.columns:
            x_unique = mushrooms_dataframe_dropped_rows[x].unique()
            print("{:>25}: {:>2} {}".format(x, x_unique.shape[0], x_unique))

        print('\n\nKodowanie danych z usuniętymi kolumnami:')
        le_cols = preprocessing.LabelEncoder()
        for column in mushrooms_dataframe_dropped_cols.columns:
            mushrooms_dataframe_dropped_cols[column] = le_cols.fit_transform(mushrooms_dataframe_dropped_cols[column])

        print("Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny po zakodowaniu:")
        for x in mushrooms_dataframe_dropped_cols.columns:
            x_unique = mushrooms_dataframe_dropped_cols[x].unique()
            print("{:>25}: {:>2} {}".format(x, x_unique.shape[0], x_unique))
```

```
Kodowanie danych z usuniętymi wierszami:
Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny po zakodowaniu:
                  classes:  2 [1 0]
                cap-shape:  6 [5 0 4 2 3 1]
              cap-surface:  4 [2 3 0 1]
                cap-color:  8 [4 7 6 3 2 5 0 1]
                  bruises:  2 [1 0]
                     odor:  7 [6 0 3 5 2 1 4]
           gill-attachment:  2 [1 0]
              gill-spacing:  2 [0 1]
                 gill-size:  2 [1 0]
                gill-color:  9 [2 3 0 4 7 1 6 5 8]
```

```
              stalk-shape:  2 [0 1]
               stalk-root:  4 [2 1 0 3]
 stalk-surface-above-ring:  4 [2 0 1 3]
 stalk-surface-below-ring:  4 [2 0 3 1]
   stalk-color-above-ring:  7 [5 2 4 3 0 1 6]
   stalk-color-below-ring:  7 [5 4 2 0 3 1 6]
              veil-color:  2 [0 1]
             ring-number:  3 [1 2 0]
               ring-type:  4 [3 0 1 2]
       spore-print-color:  6 [1 2 4 0 3 5]
              population:  6 [3 2 0 4 5 1]
                 habitat:  6 [5 1 3 0 4 2]


Kodowanie danych z usuniętymi kolumnami:
Liczba różnych wartości atrybutów wraz z ich wartościami dla każdej kolumny po zakodowaniu:
                 classes:  2 [1 0]
               cap-shape:  6 [5 0 4 2 3 1]
             cap-surface:  4 [2 3 0 1]
               cap-color: 10 [4 9 8 3 2 5 0 7 1 6]
                 bruises:  2 [1 0]
                    odor:  9 [6 0 3 5 2 1 8 7 4]
          gill-attachment:  2 [1 0]
             gill-spacing:  2 [0 1]
                gill-size:  2 [1 0]
               gill-color: 12 [ 4  5  2  7 10  3  9  1  0  8 11  6]
              stalk-shape:  2 [0 1]
 stalk-surface-above-ring:  4 [2 0 1 3]
 stalk-surface-below-ring:  4 [2 0 3 1]
   stalk-color-above-ring:  9 [7 3 6 4 0 2 5 1 8]
   stalk-color-below-ring:  9 [7 6 3 0 4 2 8 5 1]
              veil-color:  4 [2 0 1 3]
             ring-number:  3 [1 2 0]
               ring-type:  5 [4 0 2 1 3]
       spore-print-color:  9 [2 3 6 1 7 5 4 8 0]
              population:  6 [3 2 0 4 5 1]
                 habitat:  7 [5 1 3 0 4 6 2]
```

Mając zakodowane dane, należy dokonać ich podziału na atrybuty warunkowe (zmienna X_*) i decyzyjne (zmienna Y_*).

```
In [9]: X_dropped_rows = mushrooms_dataframe_dropped_rows.drop(['classes'], axis=1)
```

```
Y_dropped_rows = mushrooms_dataframe_dropped_rows['classes']

X_dropped_cols = mushrooms_dataframe_dropped_cols.drop(['classes'], axis=1)
Y_dropped_cols = mushrooms_dataframe_dropped_cols['classes']
```

Kolejnym podziałem, który należy wykonać, jest podział danych na część treningową i testową. Założono, że rozmiar części testowej będzie wynosił 33% wszystkich danych. W celu zachowania powtarzalności wyników parametr random_state ustawiono na wartość 30 (ustawienie innej wartości bedzie powodowało wygnerowanie innego podziału danych i innego drzewa decyzyjnego).

```
In [10]: random_state = 30

         X_train_dr, X_test_dr ,Y_train_dr, Y_test_dr = train_test_split(X_dropped_rows, Y_dropped_rows, test_size = 0.33, random_state=random_state)

         X_train_dc, X_test_dc ,Y_train_dc, Y_test_dc = train_test_split(X_dropped_cols, Y_dropped_cols, test_size = 0.33, random_state=random_state)
```

Utworzono funkcję sprawdzającą jakość klasyfikacji zbudowanego drzewa, funkcję budującą drzewo oraz funkcję sprawdzająca istotność atrybutu. Funkcji sprawdzająca jakość klasyfikacji zbudowanego drzewa, wypisuje wartości Accuracy, czyli współczynnik dokładności modelu do danych testowych, Precision - procent elementów będących istotnymi oraz Recall - procent istotnych elementów, które zostały wybrane.

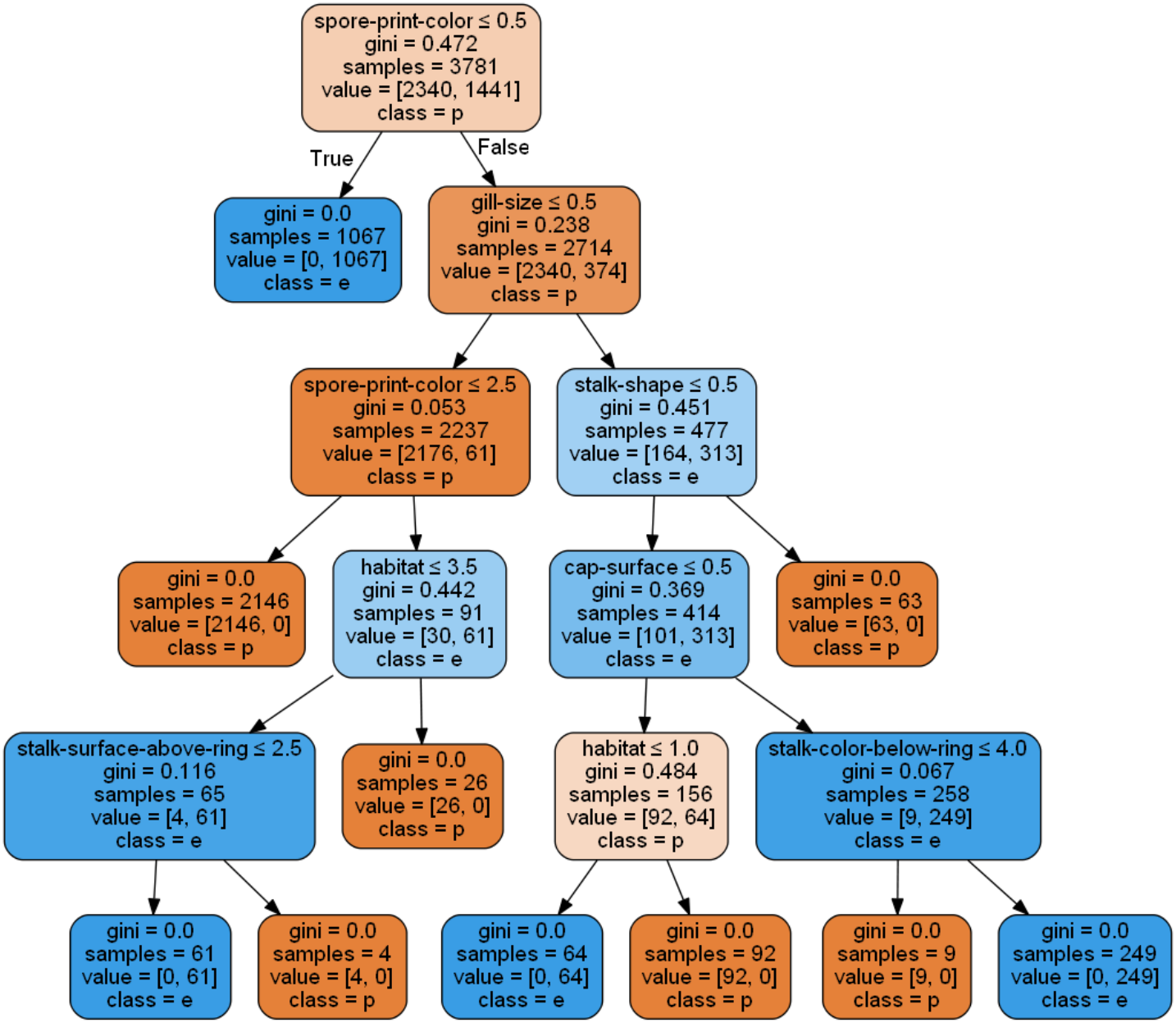
```
In [11]: def test_tree(clf, X_train, X_test, Y_train, Y_test, print_res=True):
             clf       = clf.fit(X_train, Y_train)
             score     = clf.score(X_test, Y_test)
             precision = metrics.precision_score(Y_test, clf.predict(X_test))
             recall    = metrics.recall_score(Y_test, clf.predict(X_test))
             res = (score, precision,recall)
             if print_res:
                 print("Accuracy = %f / Precision = %f / Recall = %f" % res)
             return res

         def build_tree(X, X_train, X_test, Y_train, Y_test, random_state, **kwargs):
             clf = tree.DecisionTreeClassifier(random_state=random_state, **kwargs)
             clf = clf.fit(X_train, Y_train)

             dot_data = tree.export_graphviz(clf, out_file=None,
                                             feature_names=X.columns,
                                             class_names=['p','e'],
                                             filled=True, rounded=True,
                                             special_characters=True)
             graph = pydotplus.graph_from_dot_data(dot_data)
             display(Image(graph.create_png()))

             test_tree(clf, X_train, X_test, Y_train, Y_test);
```

```
        return clf

    def attribute_importance(clf, X):
        attrs = X.columns.values
        attr_importance = clf.feature_importances_
        sorted_attr_importance = np.argsort(attr_importance)
        range_sorted_attr_importance = range(len(sorted_attr_importance))

        plt.figure(figsize=(8, 7))
        plt.barh(range_sorted_attr_importance, attr_importance[sorted_attr_importance])
        plt.yticks(range_sorted_attr_importance, attrs[sorted_attr_importance])
        plt.xlabel('Importance')
        plt.title('Attribute importances')
        plt.draw()
        plt.show()
```

W oparciu o przygotowane dane zbudowano drzewa decyzyjne.

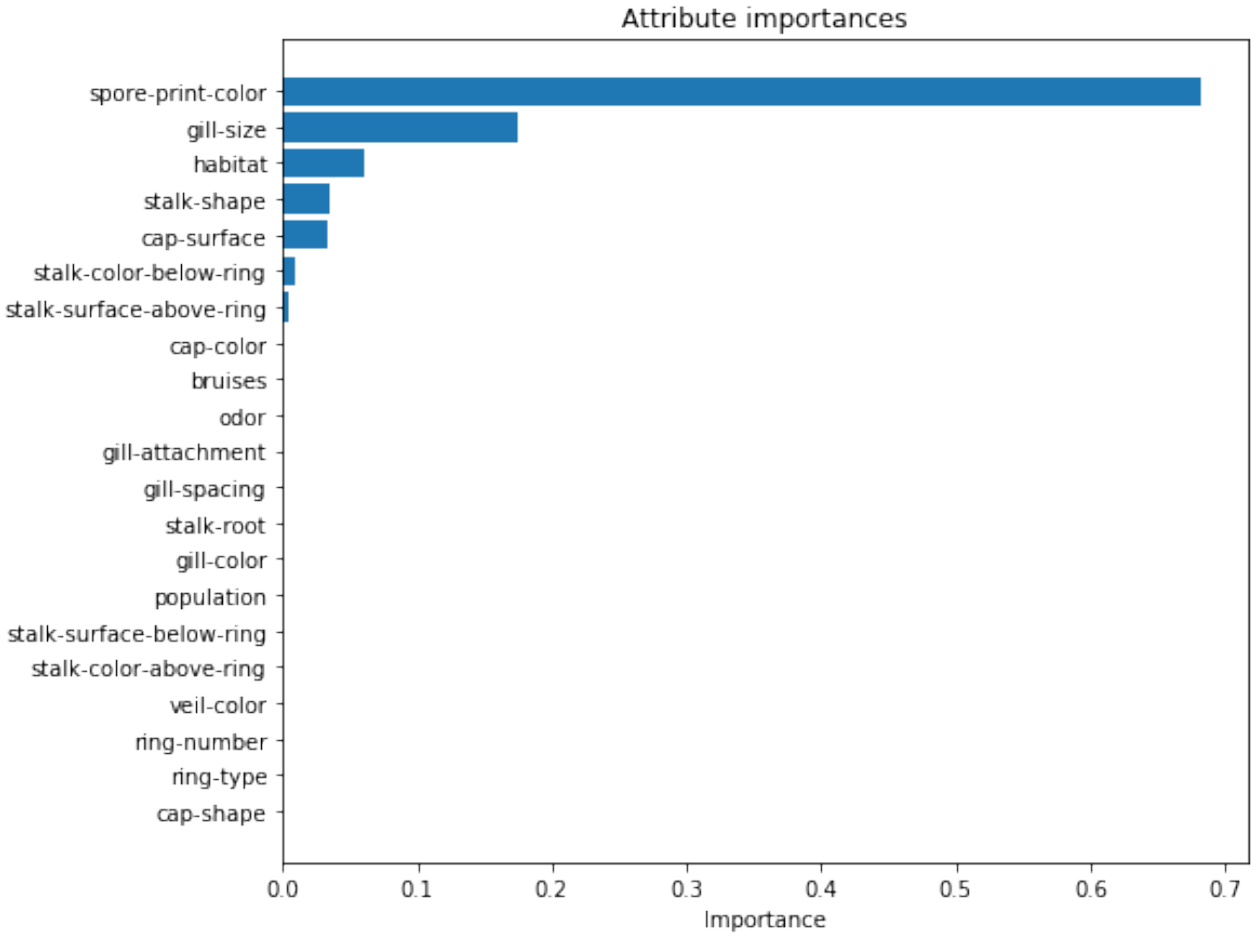
1. Dokonano klasyfikacji zbioru danych z usuniętymi wierszami, które zawierały braki danych:

```
In [12]: clf = build_tree(X_dropped_rows, X_train_dr, X_test_dr, Y_train_dr, Y_test_dr, random_state)
```



Accuracy = 1.000000 / Precision = 1.000000 / Recall = 1.000000

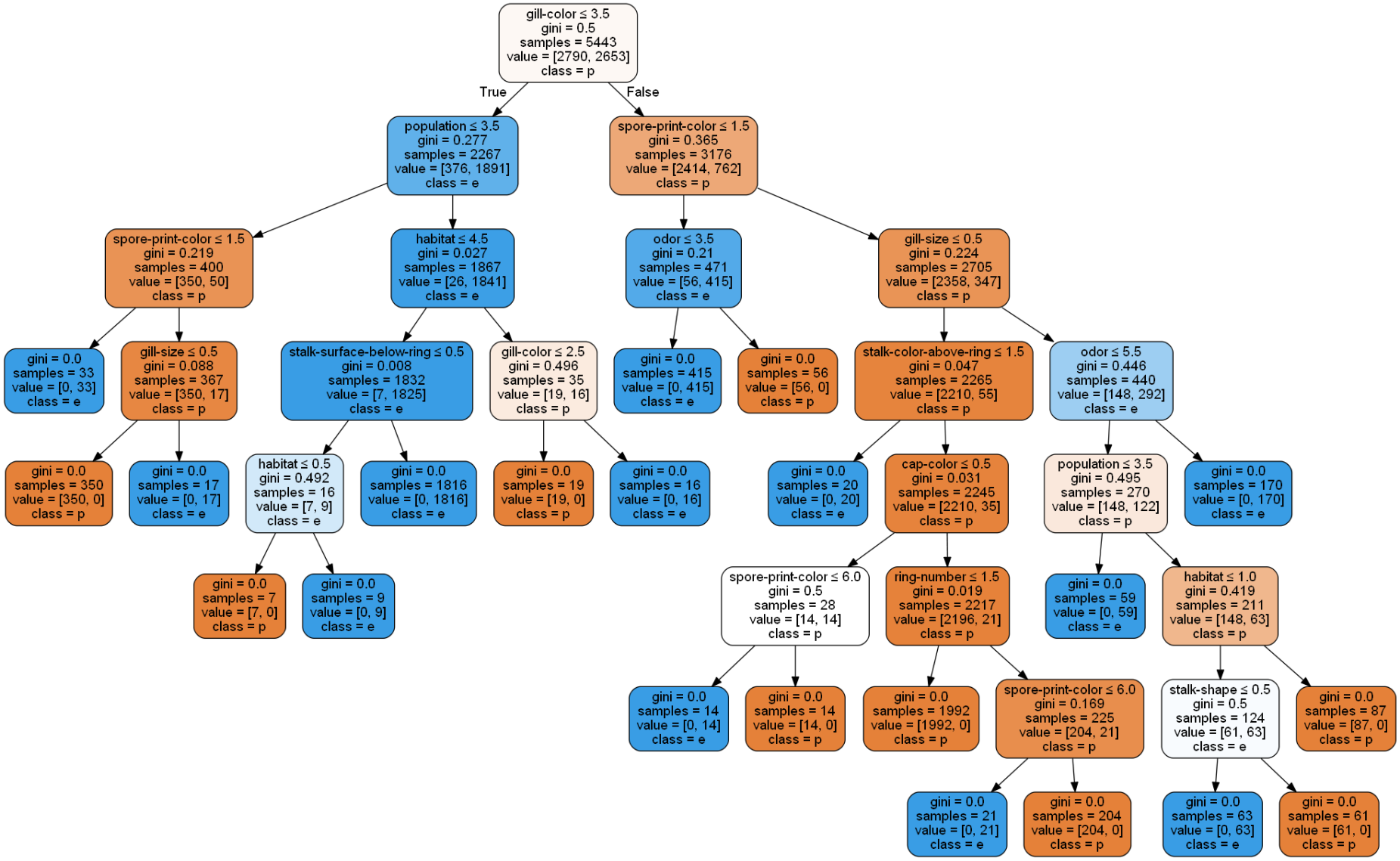
```
In [13]: attribute_importance(clf, X_dropped_rows)
```



Wygenerowane drzewo decyzyjne posiada głębokość wynoszącą 5. Ponadto zauważono, że najważnijszymi atrybutami są: spore-print-color oraz gill-size. Mniejsze znaczenie mają atrybuty: habitat, stalk-shape oraz cap-surface. Pozostałe atrybuty uznano za nieznaczące.

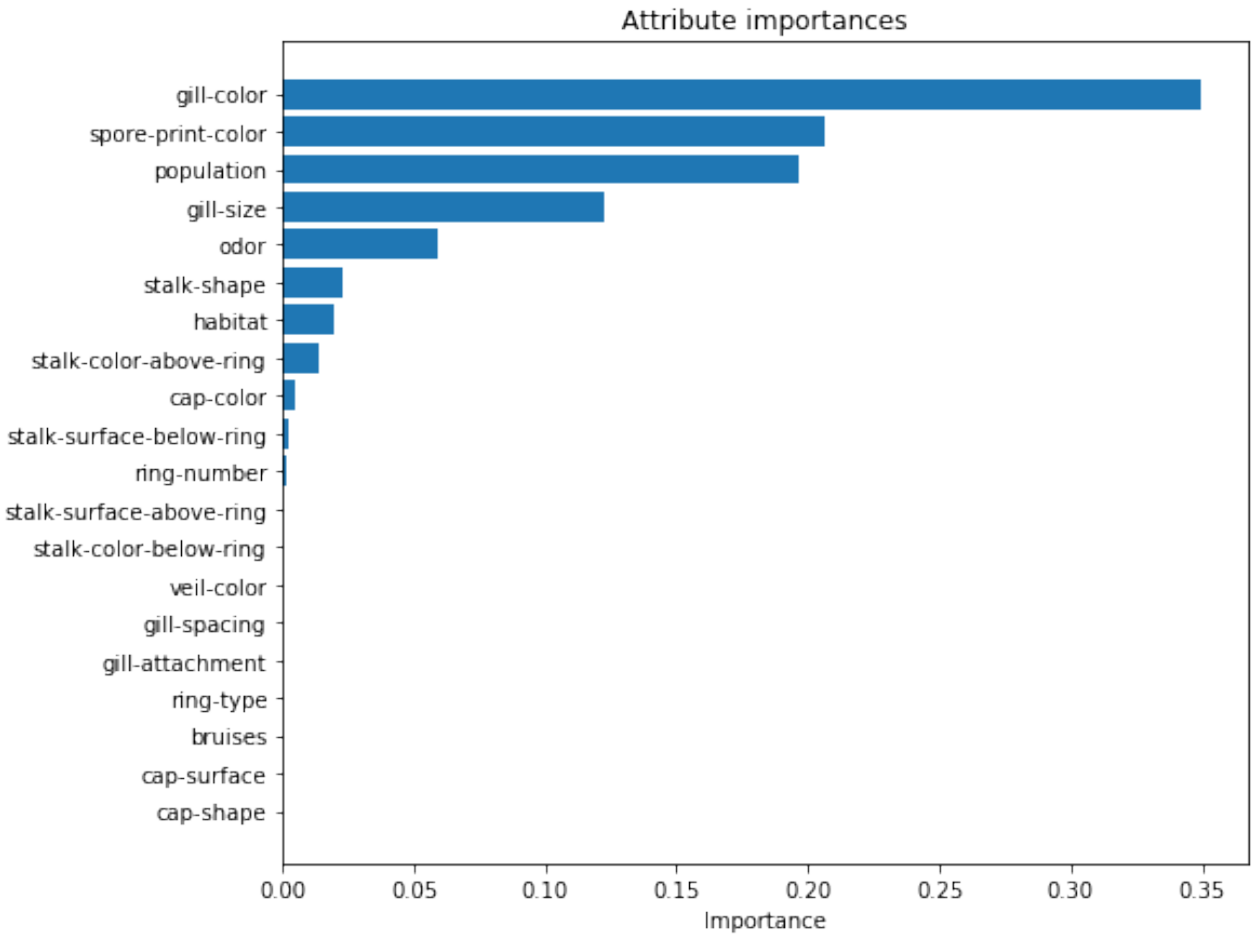
2. Dokonano klasyfikacji zbioru danych z usuniętymi kolumnami, które zawierały braki danych:

In [14]: clf = build_tree(X_dropped_cols, X_train_dc, X_test_dc ,Y_train_dc, Y_test_dc, random_state)



Accuracy = 1.000000 / Precision = 1.000000 / Recall = 1.000000

```
In [15]: attribute_importance(clf, X_dropped_cols)
```



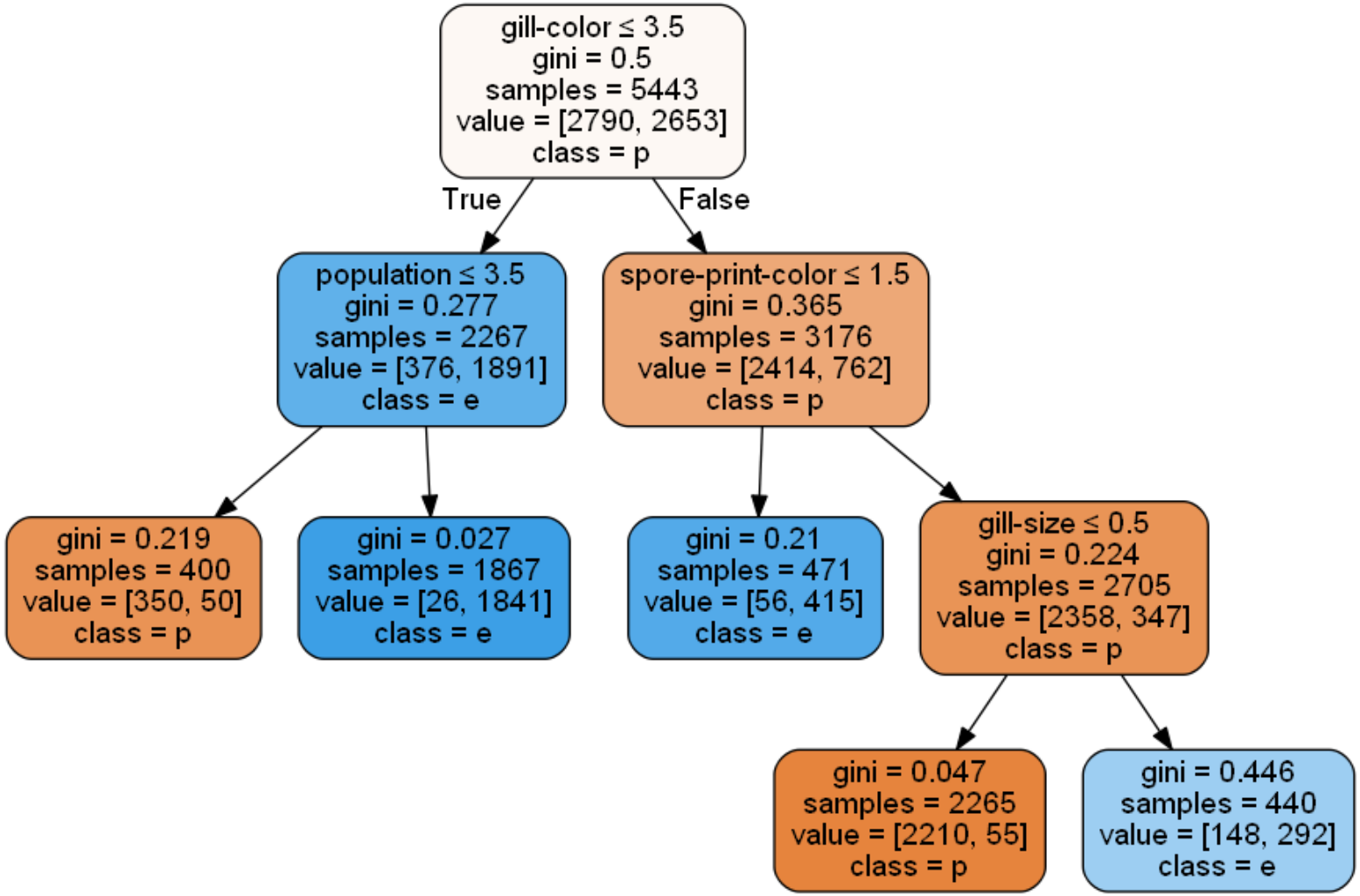
Zauważono, że drzewo wygenerowane przy użyciu danych z usuniętymi kolumnami jest bardziej rozbudowano niż drzewo wygenerowane przy użyciu danych z usuniętymi wierszami, ponieważ zostało ono skonstruowano przy użyciu zbioru o większej liczbie próbek. Wygenerowane drzewo decyzyjne posiada głębokość wynoszącą 7. Zauważono również, że najważnijszymi atrybutami są: gill-color, spore-print-color, population oraz gill-size. Mniejsze znaczenie mają atrybuty: odor, stalk-shape, habitat oraz stalk-color-above-ring. Pozostałe atrybuty uznano za nieznaczące.

Do kolejnych eksperymentów użyto z usuniętymi kolumnami, ponieważ założono że większy zbiór danych może lepiej odzwierciedlać problem klasyfikacji grzybów.

3. Dokonano klasyfikacji zbioru danych z usuniętymi kolumnami ze zmienionymi parametrami. Parametry dobierano tak, aby zmnijeszyć wielkość drzewa decyzyjnego,

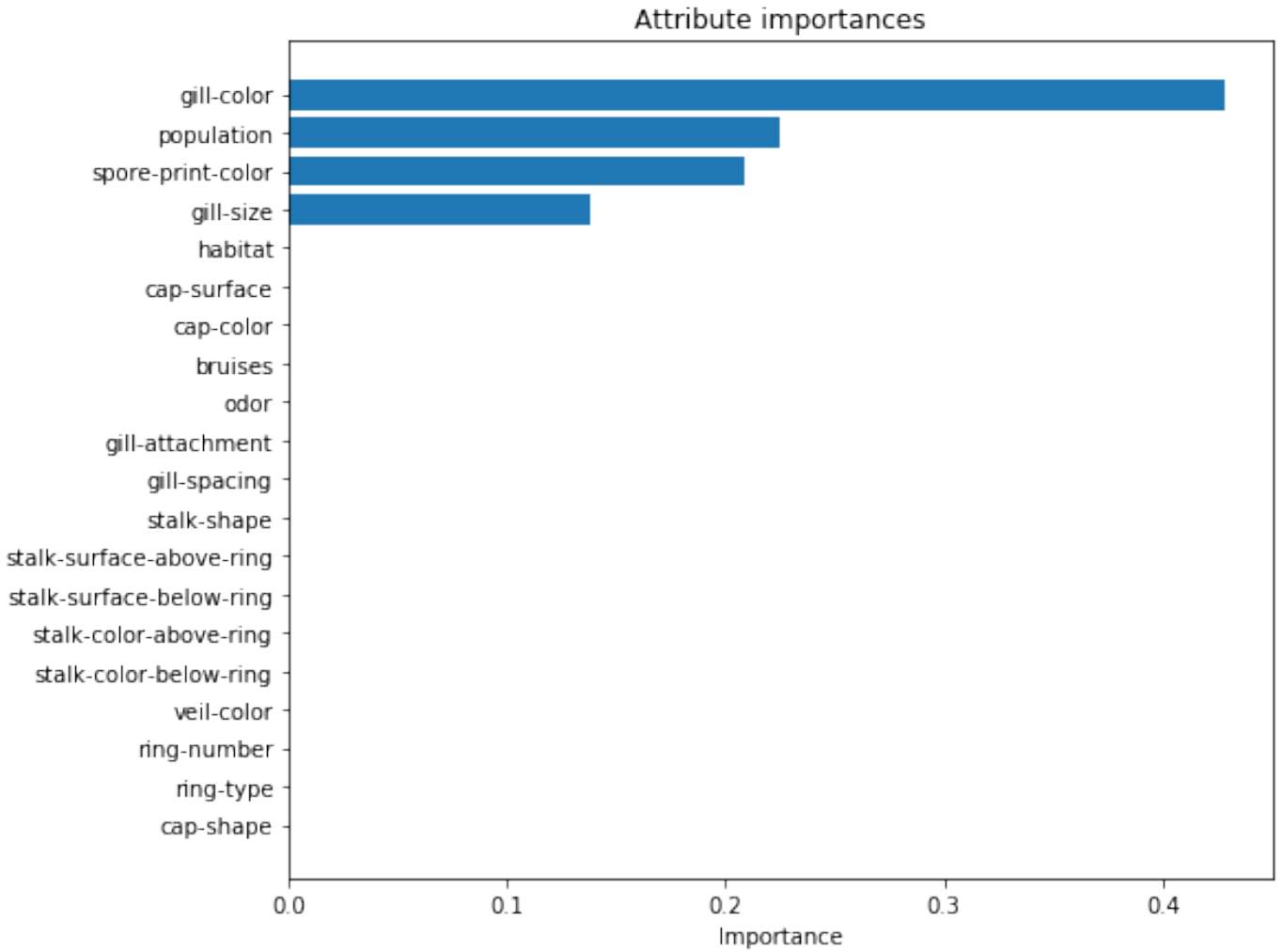
jednocześnie zachowując wysoką jakość klasyfikacji. Podczas ekspoerymentów zauważono, że w przypadku analizowanego zbioru dancych kryterium podziału nie ma istotnego wpływu na wygląd i jakość drzewa.

```
In [16]: clf = build_tree(X_dropped_cols, X_train_dc, X_test_dc ,Y_train_dc, Y_test_dc, random_state,
                          criterion="gini", max_depth=2, min_samples_split=200, min_samples_leaf=60, max_leaf_nodes=5)
```




```
Accuracy = 0.941440 / Precision = 0.919575 / Recall = 0.959620
```

```
In [17]: attribute_importance(clf, X_dropped_cols)
```



Ustalenie własnych parametrów drzewa decyzyjnego pozwoliło zmnijeszyć jego głębokość do 3, przy jednoczesnym zachowaniu dobrej jakości klasyfikacji. Współczynnik dokładności zmnijeszym się jedynie do około 94%, wartość Precission do około 91%, a Recall do około 95%. Zauważono również, że najważnijesze atrybutty, tj. gill-color, spore-print-color, population oraz gill-size zostały zachowane.

11 Wnioski

- Biblioteka Scikit-learn w łatwy sposób pozwala budować i wizualizować drzewa decyzyjne.
- Klasa DecisionTreeClassifier i DecisionTreeRegressor nie obsługuje bezpośrednio właściwości kategorycznych. Można jednak dokonać ich przekształcenia.
- Biblioteka Scikit-learn nie realizuje przycinania drzew automatycznie. Aby uniknąć przeuczeń, należy kontrolować rozmiar drzewa parametrami, np. min_samples_leaf ,min_samples_split i max_depth.
- W zależności od typu zmiennej wyjściowej należy dobrać odpowiednią klasę budującą drzewo decyzyjne - DecisionTreeClassifier w przypadku zmiennej dyskretnej, a DecisionTreeRegressor w przypadku zmiennej ciągłej.
- Drzewa decyzyjne to metody klasyfikacji, które są w stanie wyodrębnić proste reguły dotyczące cech danych, które są wnioskowane na podstawie danych wejściowych zbiór danych. Pozwalają one na odszukanie ukrytej zależności w danych, jak i również na redukcję niepotrzebnych zmiennych.
- Drzewa decyzyjne mogą być używane z wieloma zmiennymi.
- Wynik klasyfikacji, tj. liczba poprawnie przewidzianych przypadków zależy od podziału zbioru na dane treningowe i testowe oraz od parametrów budowania drzewa.
- Odpowiednie dobranie wartości parametrów drzewa, np. maximum depth pozwala zmniejszyć liczbę węzłów i liczbę podziałów przy jednoczesnym zachowaniu poprawności przewidywania. Z kolei złe ich dobranie może skutkować zmniejszeniem poprawności przewidywania przypadków w drzewie decyzyjnym.
- Jednoczesne ustawienie kilku parametrów może pozwolić na większe zredukowanie liczby węzłów, liczby podziałów i wysokości drzewa.
- Redukcja liczby węzłów lub głębokości drzewa decyzyjnego pozwala zaoszczędzić czas potrzeby na otrzymanie wyniku. Im głębsze drzewo, tym bardziej złożone staje się jego przewidywanie.
- Walidacja krzyżowa może posłużyć do oszacowania błędu i uniknięcia przejęcia.
- Przed przystąpieniem do uczenia drzewa decyzyjnego należy zbadać zbiór danych i poddać go wstępnej obróbce w celu np. usunięcia braków danych.
- Najważniejszymi atrybutami wyodrębnionymi w zbiorze ForestFires są: month oraz temp. Oznacza to, że pożary zależą najbardziej od miesiąca i temperatury.
- W zbiorze Mushroom najważniejsze atrybuty to: gill-color, spore-print-color, population oraz gill-size. Oznacza to, że klasyfikacja grzybów na trujące i jadalne zależy najbardziej od koloru blaszek, koloru wysypu zarodników, rodzaju populacji i rozmiaru blaszek.

12 Podział pracy

Podział pracy w projekcie był równomierny. Adam Bajguz przygotował program do zbioru Mushroom, a Magdalena Kalisz program do zbioru ForestFires. Dokladny podział zadań pokazano w Tabeli 2.

Tab. 2: Podział pracy

	Osoba	
Czynność lub rozdział w sprawozdaniu	**Adam Bajguz**	**Magdalena Kalisz**
Przygotowanie pliku sprawozdania w programie LaTeX	+	
Wstęp	+	
Drzewa decyzyjne w teorii decyzji i uczeniu maszynowym		+
Rodzaje drzew decyzyjnych		+
Budowa drzew decyzyjnych	+	
Idea drzew decyzji		+
Cechy drzew decyzyjnych	+	+
Rodzaje kryteriów podziału	+	
Algorytmy budowania drzew decyzyjnych		+
Przycinanie drzew decyzyjnych	+	+
Biblioteka scikit-learn: Wstęp	+	
Biblioteka scikit-learn: Klasa DecisionTreeClassifier	+	
Biblioteka scikit-learn: Klasa DecisionTreeRegressor		+
Opis wykorzystanych zbiorów danych: Forest Fires		+
Opis wykorzystanych zbiorów danych: Mushrooms	+	
Opis programów i wyników + kod: Forest Fires		+
Opis programów i wyników + kod: Mushroom	+	
Wnioski	+	+

Literatura

[1] B. W., Matching and prediction on the principle of biological classification, JRSS, Series C, Applied Statistics 8 (2).

[2] J. Morgan, J. Sonquist, Problems in the analysis of survey data, and a proposal, J. Amer. Statist. Assoc. 58 (1963) 415–434.

[3] M. P., Decision trees.
URL http://www.shogun-toolbox.org/static/notebook/current/DecisionTrees.html

[4] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106. doi:10.1007/bf00116251.
URL https://doi.org/10.1007/bf00116251

[5] J. Quinlan, Decision trees and decision-making, IEEE Transactions on Systems, Man, and Cybernetics 20 (2) (1990) 339–346. doi:10.1109/21.52545.
URL https://doi.org/10.1109/21.52545

[6] B. L., F. J., O. R., S. C., Classification and Regression Trees, Taylor & Francis Ltd, Wadsworth, Belmont, CA, 1984.

[7] G. E., Symboliczne metody klasyfikacji danych, PWN, Warszawa, 1998.

[8] A. V. C. Team, A complete tutorial on tree based modeling from scratch (in r & python) (2016).
URL https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

[9] L. P., P.-P. G., T. R., Metody sztucznej inteligencji i ich zastosowania w ekonomii i zarządzaniu, Wydawnictwo Akademii Ekonomicznej, Kraków, 2007.

[10] L. H., Decision trees and random forests for classification and regression pt.1.
URL https://towardsdatascience.com/decision-trees-and-random-forests-for-classification-and-regression-pt-1-dbb65a458df

[11] AIspace, Tutorial 4: Splitting algorithms.
URL http://aispace.org/dTree/help/tutorial4.shtml

[12] B. J., How to implement the decision tree algorithm from scratch in python (2016).
URL https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

[13] Decision tree applications for data modelling (artificial intelligence).
URL http://what-when-how.com/artificial-intelligence/decision-tree-applications-for-data-modelling-artificial-intelligence/

[14] G. V. Kass, An exploratory technique for investigating large quantities of categorical data, Applied Statistics 29 (2) (1980) 119. doi:10.2307/2986296.
URL https://doi.org/10.2307/2986296

[15] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[16] scikit learn, Decision Trees.
URL http://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart

[17] Forest fires data set, paulo Cortez and Aníbal Morais, Department of Information Systems, University of Minho, Portugal.
URL http://archive.ics.uci.edu/ml/datasets/forest+fires

[18] Mushroom data set, The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf.
URL https://archive.ics.uci.edu/ml/datasets/mushroom