

# UCZENIE SIĘ MASZYN

Drzewa i lasy losowe  
Dokumentacja końcowa

Autor: Krzysztof Marcinek  
Prowadzący: Paweł Cichosz

## 1. Wprowadzenie

Drzewa decyzyjne są jedną z najbardziej skutecznych i najpopularniejszych metod klasyfikacji. Niniejsza praca skupia się na problemie implementacji oraz testowania rodzin klasyfikatorów jakimi są lasy losowe. W celach porównawczych zaimplementowana została także możliwość konstruowania lasów deterministycznych, chociaż metoda bagging zastosowana do tworzenia podprób trenujących dla poszczególnych klasyfikatorów także i w tym wypadku wprowadza element losowości. Wszystkie niezbędne podstawy teoretyczne potrzebne do pomyślniej implementacji projektu zawarte zostały w dokumentacji wstępnej. Zawarta tam ogólna wiedza na temat rodzin klasyfikatorów została uszczegółowiona podczas implementacji projektu i będzie omawiana na bieżąco.

## 2. Implementacja

Projekt został napisany przy użyciu języka C++ w środowisku Dev-C++.

### 1. Struktury danych

Pierwszym podejściem do implementacji projektu było stworzenie klas odpowiedzialnych za przechowywanie i przetwarzania danych testowych jak i trenujących. Tak utworzone narzędzie pozwoliło sprawnie poruszać się po wczytanych zasobach ułatwiając dalszą pracę.

- **Klasa dataline**

Przechowuje pojedynczy rekord danych wraz z własnościami charakterystycznymi dla reprezentowaną przez niego bazę danych.

```
class data_line{
private:
    string tok_string;
    vector<string> attr;
    int size;
    int continuous;
    int class_column;
public:
    data_line();
    data_line(int value);
    string get_tok();
    void insert_line(string line, string tok_string, int cont, int class_column);
    string get_attr(int i);
    void set_attr(int i, string new_attr);
    int get_size();
    int if_cont();
    void erase_attr(int attr);
    void add_attr(float var);
    void set_class_column(int i);
    int get_class_column();
    void recover_attr(vector< vector<float> > array);
};
```

## ○ **Atrybuty**

string tok_string	-	string rozdzielający atrybuty
vector<string> attr	-	przechowuje wartości atrybutów
int size	-	liczba atrybutów
int continuous	-	czy atrybuty są ciągłe
int class_column	-	numer atrybuty kategorii

## ○ **Metody**

data_line()	-	konstruktor domyślny
data_line(int value)	-	konstruktor alokujący miejsce na 'value' atrybutów
string get_tok()	-	zwraca string rozdzielający atrybuty
void insert_line(string line, string tok_string, int cont, int class_column)	-	uzupełnia wcześniej zaalokowane miejsce wartościami atrybutów z wiersza 'line'
string get_attr(int i)	-	zwraca atrybut o indeksie 'i'
void set_attr(int i, string new_attr)	-	ustawia atrybut o indeksie 'i' wartością 'new_attr'
int get_size()	-	zwraca ilość atrybutów
int if_cont()	-	zwraca 1 lub 0 w zależności czy dane są ciągłe czy nie
void erase_attr(int attr)	-	usuwa atrybut o indeksie 'attr'
void add_attr(float var)	-	dodaje nowy atrybut o wartości 'var'
void set_class_column(int i)	-	ustawia atrybut o numerze i jako atrybut kategorii
int get_class_column()	-	zwraca numer atrybuty kategorii
void recover_attr (vector< vector<float> > array)	-	odtworza brakujące atrybuty z tablicy wartości współczynników; funkcja użyteczna przy algorytmie RC

- **Klasa data\_class**

Przechowuje i zarządza całą bazą danych rekordów dotyczących badanego zjawiska. Zawiera wiele funkcji użytecznych do rozwiązania problemu budowy i testowania drzew i lasów losowych jak i deterministycznych.

```
class data_class{
private:
    char* source_file;
    vector<data_line> data;
    vector<string> classes;
    vector<int> classes_inst;
    int attr_number;
    int inst_number;
    int class_number;
    int class_column;
    int continuous;
    string undefined;
    int new_attr;
    vector< vector<float> > n_attr;
    int random;
public:
    data_class(string source_file, int c, string tok_string, string undef, int contin,
               int random);
    data_class();
    string get_info();
    string get_inst(int i);
    string get_attr(int i, int j);
    void set_attr(int i, int j, string new_attr);
    string get_tok();
    string get_undef();
    int get_class_number();
    int get_inst_number();
    int get_class_column();
    int get_attr_number();
    int get_new_attr();
    int if_cont();
    int if_random();
    vector<string> get_attr_vars(int attr);
    vector<int> get_attr_vars_inst(int attr);
    string get_max_attr(int attr);
    void remove_missing(int attr);
    vector<int> random_vector(int size, int mod, int ret);
    void add_noise(float rate);
    void make_class_order();
    void make_attr_order(int attr, int left, int right);
    data_line get_data(int i);
    void erase_attr(int attr);
    void erase_line(int line);
    vector< vector<float> > add_attr(float rate);
    void recover_attr(vector< vector<float> > array);
    void make_tr_test();
};
```

## ○ **Atrybuty**

char* source_file;	-	ścieżka dostępu do pliku z danymi
vector<data_line> data	-	tablica obiektów klasy 'dataline'
vector<string> classes	-	tablica wartości atrybutu kategorii
vector<int> classes_inst	-	tablica zawierająca liczbową wartość ilości przykładów należących do danej kategorii
int attr_number	-	liczba atrybutów
int inst_number	-	liczba rekordów w bazie
int class_number	-	liczba wartości atrybutu kategorii
int class_column	-	numer atrybutu kategorii
int continuous	-	czy atrybuty są ciągłe
string undefined	-	znak reprezentujący brakujący atrybut
int new_attr	-	liczba nowych atrybutów utworzona algorytmem RC
int random	-	czy budowane drzewo ma być losowe
vector< vector<float> > n_attr	-	tablica współczynników służąca do wyznaczenia nowych atrybutów dla algorytmu RC

## ○ **Metody**

data_class(string source_file, int c, string tok_string, string undef, int contin, int random)	-	wczytuje z pliku 'source_file' bazę danych; ustawia przy tym atrybut klasy na 'c', string rozdzielający atrybuty na 'tok_string', znak brakującego atrybutu na 'undef', czy dane są ciągłe określa 'contin', zaś czy budowane drzewo ma być losowe determinuje wartość parametru 'random'
data_class()	-	konstruktor domyślny
string get_info()	-	zwraca podstawowe informacje na temat wczytaj bazy danych
string get_inst(int i)	-	wzraca wiersz danych o indeksie 'i' w postaci stringu
string get_attr(int i, int j)	-	wzraca atrybut w wierszu 'i' o indeksie 'j'
void set_attr(int i, int j, string new_attr)	-	ustawia atrybut w wierszu 'i' o indeksie 'j' wartości 'new_attr'
string get_tok()	-	zwraca znak rozdzielający atrybuty
string get_undef()	-	zwraca znak reprezentujący brakujący atrybut
int get_class_number()	-	zwraca numer atrybuty kategorii
int get_inst_number()	-	zwraca ilczbę rekordów danych
int get_class_column()	-	zwraca numer atrybuty kategorii

<code>int get_attr_number()</code>	-	zwraca ilość atrybutów
<code>int get_new_attr()</code>	-	zwraca ilość nowych atrybutów utworzonych algorytmem RC
<code>int if_cont()</code>	-	zwraca 1 lub 0 w zależności czy dane są ciągłe czy nie
<code>int if_random()</code>	-	czy budowane drzewo ma być losowe
<code>vector&lt;string&gt; get_attr_vars(int attr)</code>	-	zwraca wartości atrybutu o indeksie 'attr'
<code>vector&lt;int&gt; get_attr_vars_inst(int attr)</code>	-	zwraca ilość rekordów mających daną wartość atrybutu o indeksie 'attr'
<code>string get_max_attr (int attr)</code>	-	zwraca najczęściej występującą wartość atrybutu o indeksie 'attr'
<code>void remove_missing (int attr)</code>	-	zastępuje brakujące atrybuty o indeksie 'attr' najczęściej występującymi wartościami
<code>vector&lt;int&gt; random_vector(int size, int mod, int ret)</code>	-	generuje wektor o długości 'size' losowych wartości modulo 'mod' ze zwracaniem lub bez (ret = 1/0)
<code>void add_noise(float rate)</code>	-	zamienia wartość atrybutu kategorii wartością losową ze zbioru wartości atrybutu klasy w losowym wektorze numerów rekordów o długości, której procentową wartość określa parametr 'rate'
<code>void make_class_order()</code>	-	przenosi atrybut kategorii na koniec wiersza danych
<code>void make_attr_order (int attr, int left, int right)</code>	-	funkcja implementująca algorytm Q-sort do porządkowania rekordów według wartości atrybutu 'attr'
<code>data_line get_data(int i)</code>	-	zwraca obiekt klasy 'data_line' będący 'i'-tym rekordem danych
<code>void erase_attr(int attr)</code>	-	usuwa ze wszystkich rekordów atrybut o indeksie 'attr'
<code>void erase_line(int line)</code>	-	usuwa rekord o indeksie 'line'
<code>vector&lt; vector&lt;float&gt; &gt; add_attr(float rate)</code>	-	zwraca tablicę wartości losowych rozłożonych równomiernie w przedziale [-1;1]; wymiary tablicy wynoszą 'attr_numer' x m, gdzie m = ceil('rate'*'attr_numer'); na podstawie utworzonej tablicy dla każdego rekordu generuje m nowych atrybutów

<code>void recover_attr (vector&lt; vector&lt;float&gt; &gt; array)</code>	-	dokonyuje odtworzenia brakujących atrybutów na podstawie tablicy 'array'
<code>void make_tr_test()</code>	-	tworzy dwa pliki zawierające losowe wektory danych z celu stworzenia zbiorów trenującego i testowego; podział: trenujący – 75%, testowy - 25%

## 2. Tworzenie drzew i lasów

Po zaimplementowanie struktur danych dalsza część pracy polegała na stworzeniu narzędzi służących do budowy drzew i lasów z wykorzystaniem wcześniej napisanych klas.

### ● Klasa tree

Tworzy pojedyncze drzewo na podstawie danych zawartych w obiekcie klasy 'data\_class'.

```
class tree{
private:
    vector<tree> array;
    vector<string> test_vars;
    vector<int> to_del;
    string max_var;
    int test;
    string class_;
    vector< vector<float> > RC;
public:
    tree(data_class data_array);
    string find_class(data_line data);
    vector< vector<float> > get_rc();
    void set_rc(vector< vector<float> > rc);
};
```

#### ○ Atrybuty

<code>vector&lt;tree&gt; array</code>	-	przechowuje kolejne węzły drzewa
<code>vector&lt;string&gt; test_vars</code>	-	przechowuje wartości atrybutu na podstawie którego wykonany był podział
<code>vector&lt;int&gt; to_del</code>	-	wektor atrybutów pominiętych podczas tworzenia drzewa
<code>string max_var</code>	-	najczęściej występująca wartość atrybuty podziału
<code>int test</code>	-	numer atrybutu podziału
<code>string class_</code>	-	końcowa wartość klasyfikacji drzewa
<code>vector&lt; vector&lt;float&gt; &gt; RC</code>	-	zwraca tablicę wartości losowych rozłożonych równomiernie w przedziale [-1;1]; na jej podstawie dokonywana jest regeneracja atrybutów w algorytmie RC

- **Metody**

<code>tree(data_class data_array)</code>	-	tworzy drzewo na podstawie obiektu klasy 'data_class'
<code>string find_class(data_line data)</code>	-	wyszukuje w drzewie kategorię przykładu testowego na podstawie atrybutów zawartych w obiekcie klasy 'data_line'
<code>vector&lt; vector&lt;float&gt; &gt; get_rc()</code>	-	zwraca tablicę współczynników algorytmu RC
<code>void set_rc(vector &lt; vector&lt;float&gt; &gt; rc)</code>	-	ustawia tablicę współczynników algorytmu RC

- **Klasa forest**

Tworzy las drzew jako tablicę obiektów klasy 'tree'. Docelowa klasa w hierarchii projektu.

```
class forest{  
private:  
    vector<tree> array;  
public:  
    forest(data_class training, float RC_rate, int number);  
    int get_size();  
    vector<string> find_class(data_class data);  
};
```

- **Atrybuty**

<code>vector&lt;tree&gt; array</code>	-	przechowuje korzenie wszystkich drzew w lesie
---------------------------------------	---	---

- **Metody**

<code>forest(data_class training, float RC_rate, int number)</code>	-	tworzy 'numer' drzew na podstawie danych zawartych w obiekcie 'data_class', gdzie 'RC_rate' określa ułamkowy udział nowych atrybutów
<code>int get_size()</code>	-	zwraca liczbę drzew w lesie
<code>vector&lt;string&gt; find_class (data_class data)</code>	-	wyszukuje w lesie klasy przykładów testowych na podstawie atrybutów zawartych w obiekcie klasy 'data_class'



### 3. Zasada działania algorytmów

#### ● Atrybuty

W projekcie przewidziano obsługę atrybutów ciągłych i nominalnych, przy czym w danym lasie może występować tylko jedna z tych klas atrybutów. Atrybuty numeryczne można obsługiwać zarówno jako ciągłe jak i nominalne. Różnica w sposobie budowy drzewa na podstawie klas atrybutów zostanie opisana w dalszej części dokumentacji.

#### ● Algorytmy

Głównym założeniem projektowym jest badania własności lasów losowych. W zastosowanym algorytmie na każdym poziomie budowy drzewa następuje wybór jednego losowego atrybutu podziału.

W celach porównawczych istnieje możliwość stworzenia lasu deterministycznego, który jako atrybut podziału wybiera jeden z atrybutów dających najwięcej rozgałęzień na danym poziomie.

Dodatkowo w przypadku użycia atrybutów ciągłych istnieje możliwość stworzenia szeregu nowych atrybutów jako kombinację liniową atrybutów już istniejących. Tablica współczynników zawiera losowe wartości o rozkładzie równomiernym w przedziale  $[-1;1]$ . Generacja nowych atrybutów zachodzi tylko w korzeniu każdego drzewa.

#### ● Budowa drzewa

Istnieją trzy warunki stopu budowy drzewa. Pierwszym z nich jest stwierdzenie, że wszystkie przykłady należą do jednej klasy, drugim brak przykładów, trzecim natomiast jest przypadek braku dalszych atrybutów do podziału. W trzecim przypadku kategoria tak stworzonego liścia jest wartością kategorii najczęściej występująca wśród przykładów.

Drzewo budowane jest rekurencyjnie od korzenia aż do poziomu liści. Na każdym poziomie wykonywane są następujące operacje:

- Wygenerowanie atrybutu podziału.
- W przypadku atrybutów nominalnych następuje sprawdzenie czy nie występują brakujące wartości atrybutów. Jeżeli tak, to są zastępowane najczęściej występującą wartością. Nie został zaimplementowany sposób usuwania brakujących wartości dla atrybutów ciągłych.
- Następuje podział zbioru danych ze względu na atrybut dzielący. W przypadku atrybutów nominalnych zbiorem wartości dzielących są wartości atrybutu podziału. Dane ciągłe są natomiast sortowane, a zbiór wartości dzielących stanowią liczby równoodległe od dwóch sąsiednich wartości atrybutu podziału. Tworzona jest także wartość wartownika  $+\text{inf}$ .
- W przypadku, gdy dany atrybut daje podział na mniej niż dwa podzbiory jest on usuwany ze zbioru danych i następuje generacja nowego atrybutu.
- Następnie dla każdego stworzonego podzbioru wywoływana jest funkcja budowy drzewa.

## ● Budowa lasu

Dla każdego drzewa w lesie wykonywane są następujące operacje:

- Wygenerowanie podzbioru zbioru trenującego metodą bagging. W zastosowanej implementacji jest to około połowa przykładów zbioru trenującego.
- O ile to możliwe następuje generacja nowych atrybutów.
- Na tak spreparowanym zbiorze trenujących uruchamiany jest algorytm budowy drzewa.

## ● Wyszukiwanie w drzewie

Ma na celu znalezienie kategorii na podstawie danego zbioru atrybutów. Poczynając od korzenia wykonywane są następujące operacje:

- Kryterium stopu jest znalezienie się w węźle o wartości atrybutu kategorii różnej od zbioru pustego.
- Dla każdego węzła następuje usunięcie atrybutów przykładu, które z tym węźle zostały przeznaczone do usunięcia.
- Ze zbioru atrybutów przykładu wybierany jest atrybut o numerze zapisanym w węźle drzewa jako atrybut podziału.
- W przypadku atrybutów nominalnych kolejnym węzłem jest węzeł o numerze zgodnym z położeniem w tablicy wartości atrybutu równego atrybutowi przykładu. Dla przykładów ciągłych wybierany jest węzeł o najmniejszej wartości większej od wartości atrybutu podziału.

## ● Wyszukiwanie w lesie

Znajduje wartości kategorii dla całego szeregu przykładów. Dla każdego przykładu testującego wykonywane są następujące operacje:

- Dla każdego drzewa odtwarzane są wartości nowych atrybutów na podstawie tablicy współczynników.
- Dla każdego drzewa następuje wyszukanie wartości atrybutu klasy przykładu.
- Kategorią przykładu jest najczęściej występująca odpowiedź poszczególnych drzew lasu.

## 4. Metody testowania

Dalszym krokiem po pomyślnym zaimplementowaniu algorytmów drzew jest ich testowanie. W tym celu napisana została klasa testująca.

### ● Klasa test

Posiada cztery funkcje testujące parametry zbudowanych klasyfikatorów. Każda z funkcji posiada ten sam zestaw parametrów wejściowych.

#### ○ Metody

##### ■ general

Zwraca plik o nazwie 'general\_noise\_test.txt', w którym zawarte są informacje na temat zbiorów trenujących i testowych oraz błąd klasyfikacji zbioru testowego dla klasyfikatorów złożonych z 25 drzew. Wyniki uśredniane są po wykonaniu 'runs' przebiegów.

##### ■ error\_tree\_number

Zwraca plik o nazwie 'error\_tree\_number.txt', w którym zawarte są informacje na temat błędu klasyfikacji zbudowanych klasyfikatorów w funkcji liczby drzew.

##### ■ noise

Zwraca plik o nazwie 'noise.txt', w którym zawarte są informacje na temat błędu klasyfikacji zbudowanych klasyfikatorów w funkcji wprowadzonych zakłóceń do zbioru trenującego.

##### ■ stability

Zwraca plik o nazwie 'stability.txt', w którym zawarte są informacje na temat procentowej liczby przykładów zakwalifikowanych do różnych kategorii przez parę klasyfikatorów zbudowanych na takich samych danych wejściowych w funkcji liczby drzew.

#### ○ Parametry wejściowe

string tr_file	-	ścieżka do pliku z danymi trenującymi
string test_file	-	ścieżka do pliku z danymi testującymi
int c	-	numer atrybuty kategorii
string tok_string	-	string rozdzielający atrybuty
string undef	-	znak atrybuty brakującego
int contin	-	czy dane są ciągłe
int runs	-	liczba przebiegów testujących
int number	-	liczba drzew w lesie lub procent zaszumienia
float rate	-	ułamkowy udział nowych atrybutów

## 5. Testowanie

Rozdział ten przedstawia wyniki oraz interpretację wyników uzyskanych na zsergu zestawów baz danych. Wszystkie wyniki symulacje są uśrednionymi wartościami po trzech przebiegach. Liczba drzew w klasyfikatorze równa jest 25 z wyjątkiem testów w funkcji liczby drzew.

- **general**

- **atrybuty nominalne**

- **car database**

Train:

Number of attributes: 6 + 1 class attribute

Number of instances: 1296

Number of classes: 4

unacc: 918 instances

acc: 282 instances

vgood: 48 instances

good: 48 instances

Test:

Number of attributes: 6 + 1 class attribute

Number of instances: 432

Number of classes: 4

unacc: 292 instances

acc: 102 instances

vgood: 17 instances

good: 21 instances

7.71605%

las losowy

26.6204%

drzewo losowe

24.537%

las deterministyczny

- **nursery database**

Train:

Number of attributes: 8 + 1 class attribute

Number of instances: 9720

Number of classes: 5

priority: 3136 instances

not\_recom: 3240 instances

recommend: 1 instances

very\_recom: 228 instances

spec\_prior: 3115 instances

Test:

Number of attributes: 8 + 1 class attribute

Number of instances: 3239

Number of classes: 5

spec\_prior: 928 instances

priority: 1130 instances

very\_recom: 100 instances

not\_recom: 1080 instances

recommend: 1 instances

10.9396%	las losowy
22.9906%	drzewo losowe
14.0475%	las deterministyczny

■ **audiology.standardized database (missing attributes)**

Train:

Number of attributes: 70 + 1 class attribute

Number of instances: 200

Number of classes: 24

- cochlear\_unknown: 48 instances
- mixed\_cochlear\_age\_fixation: 1 instances
- mixed\_cochlear\_age\_otitis\_media: 4 instances
- cochlear\_age: 46 instances
- normal\_ear: 20 instances
- cochlear\_poss\_noise: 16 instances
- cochlear\_age\_and\_noise: 18 instances
- acoustic\_neuroma: 1 instances
- mixed\_cochlear\_unk\_ser\_om: 3 instances
- conductive\_discontinuity: 2 instances
- retrocochlear\_unknown: 2 instances
- conductive\_fixation: 6 instances
- bells\_palsy: 1 instances
- cochlear\_noise\_and\_hereditiy: 2 instances
- mixed\_cochlear\_unk\_fixation: 5 instances
- mixed\_poss\_noise\_om: 2 instances
- otitis\_media: 4 instances
- possible\_menieres: 8 instances
- possible\_brainstem\_disorder: 4 instances
- cochlear\_age\_plus\_poss\_menieres: 1 instances
- mixed\_cochlear\_age\_s\_om: 2 instances
- mixed\_cochlear\_unk\_discontinuity: 2 instances
- mixed\_poss\_central\_om: 1 instances
- poss\_central: 1 instances

Test:

Number of attributes: 70 + 1 class attribute

Number of instances: 26

Number of classes: 6

- cochlear\_age: 11 instances
- cochlear\_age\_and\_noise: 4 instances
- cochlear\_poss\_noise: 4 instances
- mixed\_cochlear\_unk\_fixation: 4 instances
- normal\_ear: 2 instances
- mixed\_cochlear\_age\_fixation: 1 instances

28.2051%	las losowy
48.7179%	drzewo losowe
57.6923%	las deterministyczny

- **atrybuty ciągłe**

- **balance-scale database**

Train:

Number of attributes: 4 + 1 class attribute

Number of instances: 469

Number of classes: 3

B: 36 instances

R: 212 instances

L: 221 instances

Test:

Number of attributes: 4 + 1 class attribute

Number of instances: 156

Number of classes: 3

L: 67 instances

R: 76 instances

B: 13 instances

18.1624%

las losowy

30.5556%

drzewo losowe

24.359%

las deterministyczny

16.0256%

las RC

- **ionosphere database**

Train:

Number of attributes: 34 + 1 class attribute

Number of instances: 264

Number of classes: 2

g: 174 instances

b: 90 instances

Test:

Number of attributes: 34 + 1 class attribute

Number of instances: 87

Number of classes: 2

g: 51 instances

b: 36 instances

16,8582%

las losowy

24.9042%

drzewo losowe

18,0077%

las deterministyczny

16,8582%

las RC

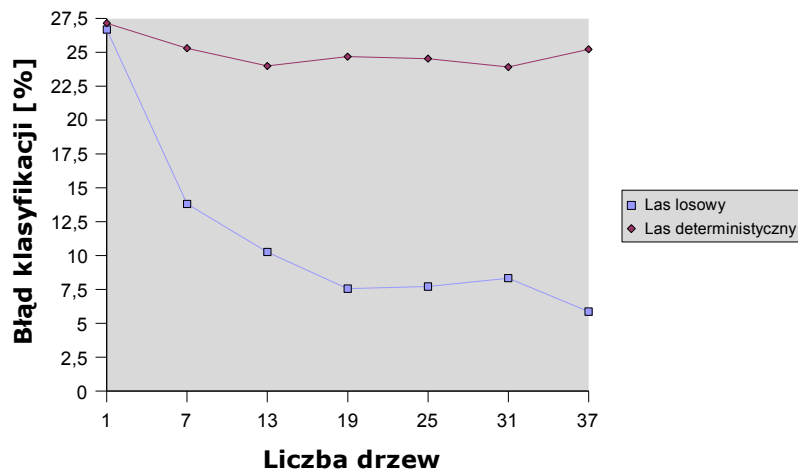
Przedstawione wyniki potwierdzają lepszą jakość rodzin klasyfikatorów nad ich pojedynczymi odpowiednikami. Lepsze wyniki wydają się prezentować także rodziny losowe. Las zbudowany algorytmem RC zgodnie z oczekiwaniami sprawuje się nieco lepiej w przypadku niewielkiej liczby atrybutów. Znacząca przewaga lasu losowego pokazana została w przypadku bazy audiology.standardized, gdzie każde nowe drzewo w lesie deterministycznym powieliło błędy poprzedniego w taki sam sposób uzupełniając brakujące atrybuty. Las losowy ze względu na sposób wyboru atrybutu podziału pozbawiony jest tego obciążenia.

- **error\_tree\_number**

- **dane testowe**

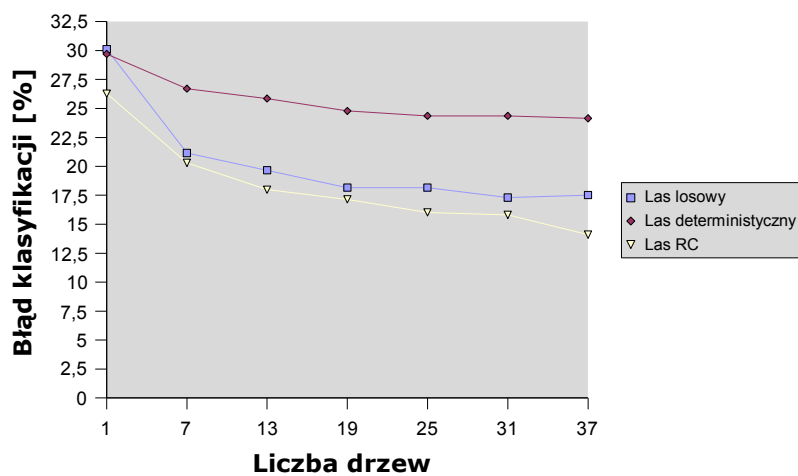
- **car database**

Błąd na danych testowych



- **balance-scale database**

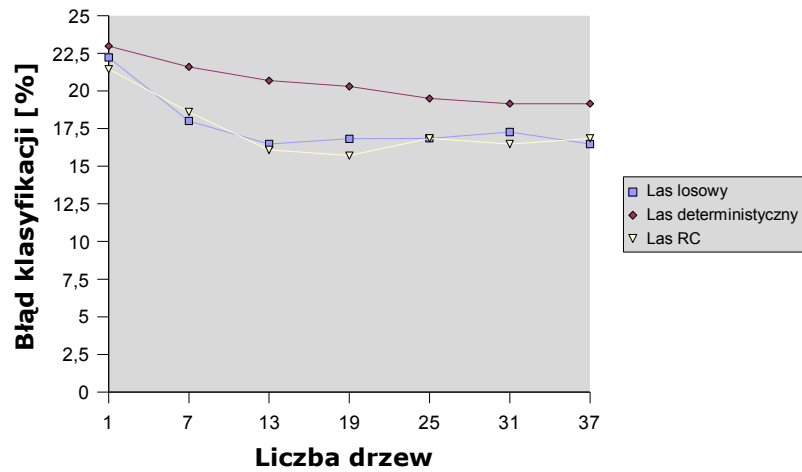
Błąd na danych testowych



L.d.	Las losowy	Las det.	Las RC
1	30.13	29.70	26.28
7	21.15	26.71	20.30
13	19.66	25.85	17.99
19	18.16	24.79	17.15
25	18.16	24.36	16.03
31	17.31	24.36	15.81
37	17.52	24.15	14.10

## ■ ionosphere database

Błąd na danych testowych

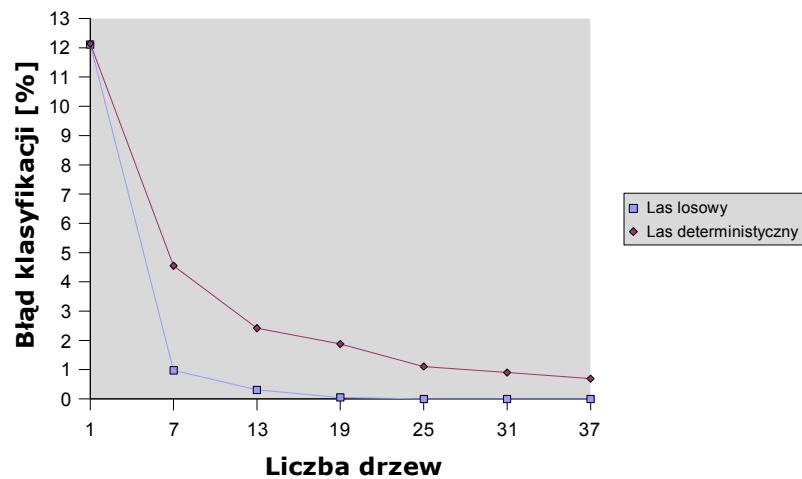


L.d.	Las losowy	Las det.	Las RC
1	22.22	22.99	21.46
7	18.01	21.61	18.61
13	16.48	20.69	16.09
19	16.83	20.31	15.71
25	16.86	19.51	16.86
31	17.27	19.16	16.48
37	16.48	19.16	16.86

## ○ dane trenujące

### ■ car database

Błąd na danych trenujących

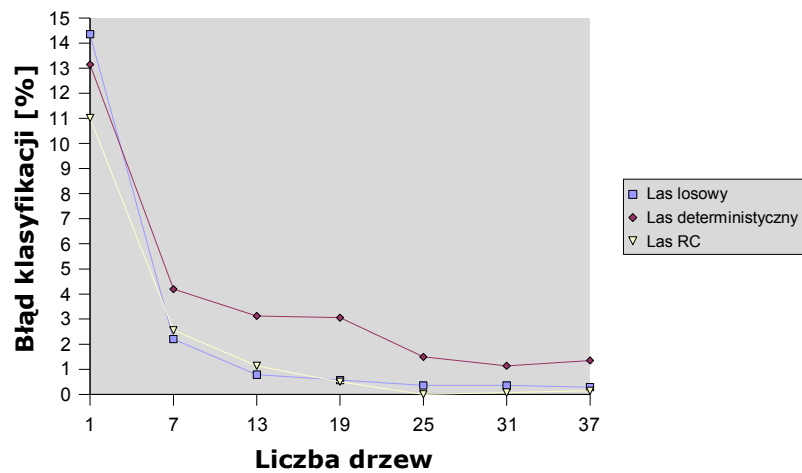


L.d.	Las losowy	Las det.
1	12.11	12.14
7	0.98	4.55
13	0.31	2.42
19	0.05	1.88
25	0.00	1.11
31	0.00	0.90
37	0.00	0.69



## ■ balance-scale database

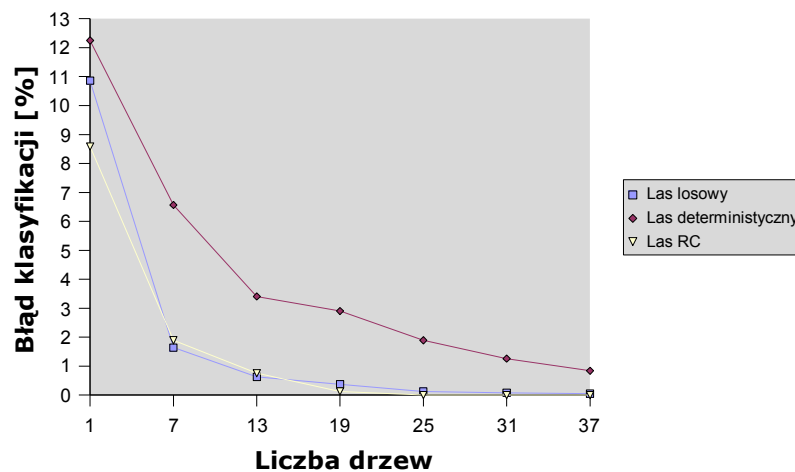
Błąd na danych trenujących



L.d.	Las losowy	Las det.	Las RC
1	14.36	13.15	11.02
7	2.20	4.19	2.56
13	0.78	3.13	1.14
19	0.57	3.06	0.50
25	0.36	1.49	0.00
31	0.36	1.14	0.07
37	0.28	1.35	0.14

## ■ ionosphere database

Błąd na danych trenujących

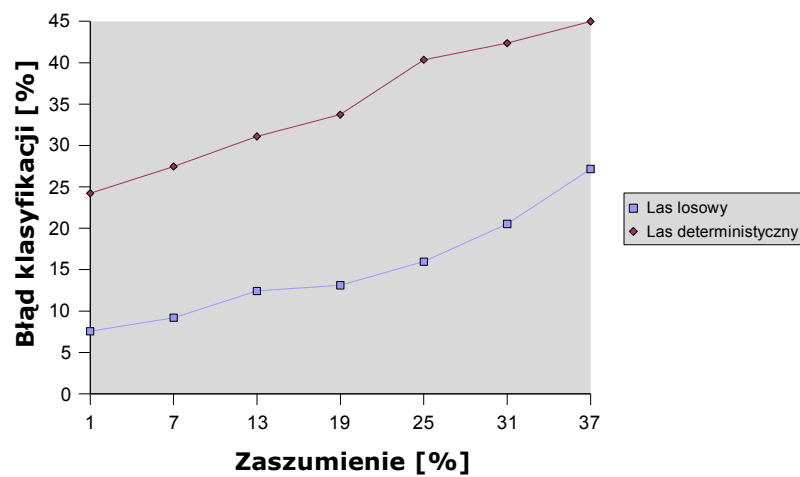


L.d.	Las losowy	Las det.	Las RC
1	10.86	12.25	8.59
7	1.64	6.57	1.89
13	0.63	3.41	0.76
19	0.38	2.90	0.13
25	0.13	1.89	0.00
31	0.08	1.26	0.00
37	0.05	0.84	0.00

- noise

- car database

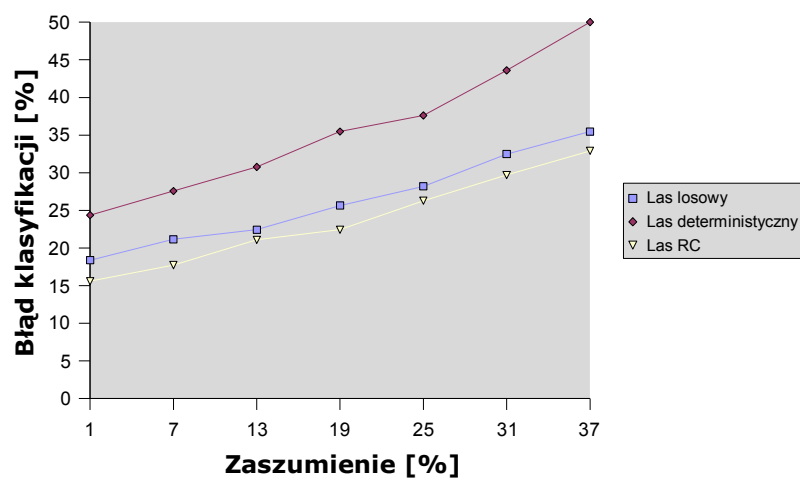
Błąd na danych testowych



L.d.	Las losowy	Las det.
1	7.56	24.23
7	9.18	27.47
13	12.42	31.10
19	13.12	33.72
25	15.97	40.35
31	20.52	42.36
37	17.16	44.98

- balance-scale database

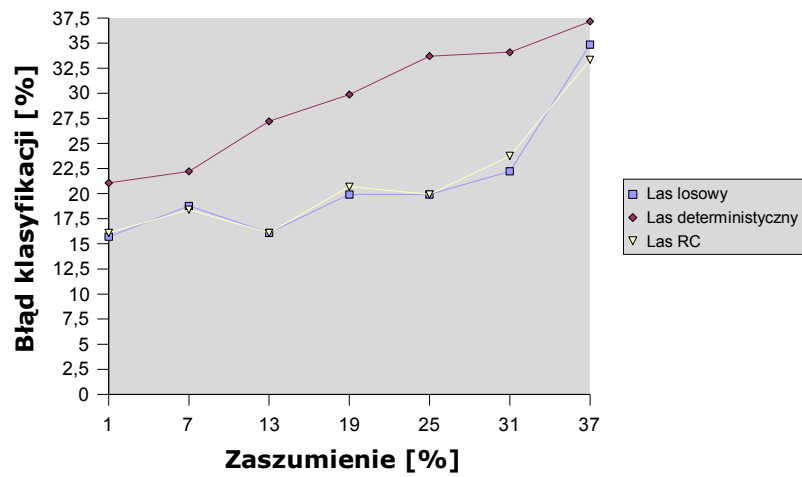
Błąd na danych testowych



L.d.	Las losowy	Las det.	Las RC
1	18.38	24.36	15.60
7	21.15	27.56	17.74
13	22.44	30.77	21.08
19	25.64	35.47	22.44
25	28.21	37.61	26.28
31	32.48	43.59	29.70
37	35.47	50.00	32.71

## ■ ionosphere database

Błąd na danych testowych

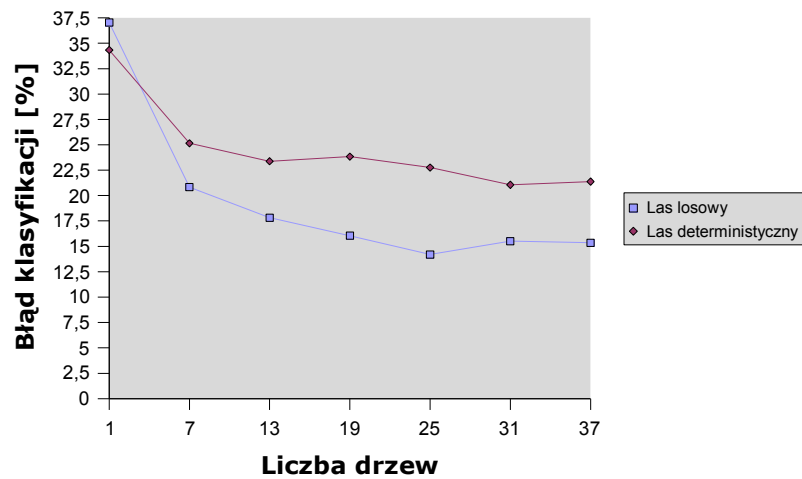


L.d.	Las losowy	Las det.	Las RC
1	15.71	21.07	16.09
7	18.77	22.22	18.39
13	16.09	27.20	16.09
19	19.92	29.89	20.69
25	19.92	33.72	19.92
31	22.22	34.10	23.75
37	34.87	37.16	33.33

## ● stability

### ■ car database

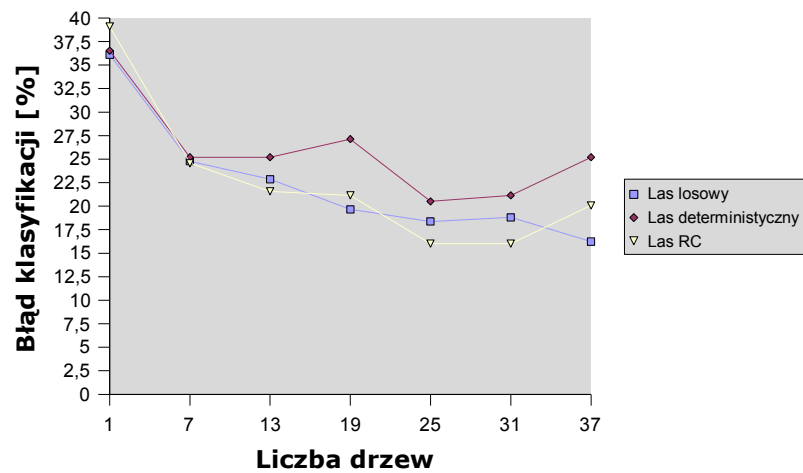
Błąd na danych testowych



L.d.	Las losowy	Las det.
1	37.04	34.34
7	20.83	25.15
13	17.82	23.38
19	16.05	23.84
25	14.20	22.76
31	15.51	21.06
37	15.35	21.37

## ■ balance-scale database

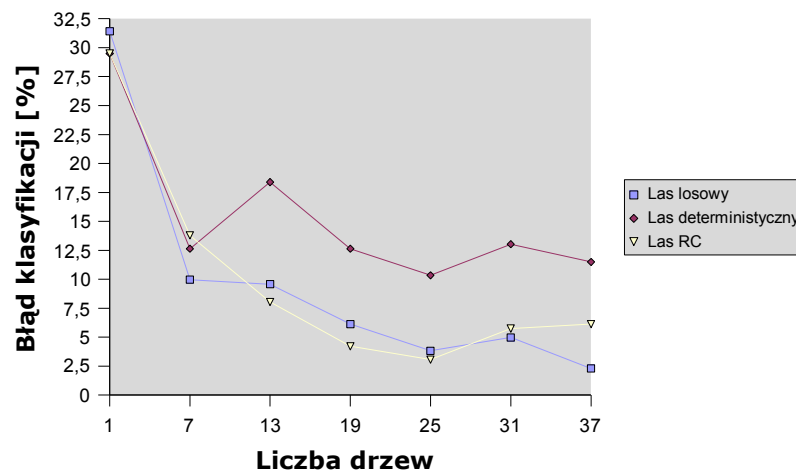
Błąd na danych testowych



L.d.	Las losowy	Las det.	Las RC
1	36.11	36.54	39.10
7	24.79	25.21	24.57
13	22.86	25.21	21.58
19	19.66	27.14	21.15
25	18.38	20.51	16.03
31	18.80	21.15	16.03
37	16.24	25.21	20.09

## ■ ionosphere database

Błąd na danych testowych



L.d.	Las losowy	Las det.	Las RC
1	31.42	29.50	29.50
7	9.96	12.64	13.79
13	9.58	18.39	8.05
19	6.13	12.64	4.21
25	3.83	10.34	3.07
31	4.98	13.03	5.75
37	2.30	11.49	6.13

## 6. Wnioski

Zgodnie z oczekiwaniami błąd klasyfikacji lasów maleje wraz ze zwiększającą się liczbą drzew. W przypadku lasu deterministycznego występuje jednak pewien poziom nasycenia. Spowodowane jest to rosnącym współczynnikiem korelacji między drzewami w lesie.

Wykresy przedstawiające błąd na danych trenujących mają na celu pokazanie charakteru dopasowania się klasyfikatora do danych trenujących. w tym miejscu można zadać pytanie o zjawisko przeuczenia. Analizując jednak otrzymane wykresy można zauważyć, że gdy błąd na danych trenujących maleje, takie same tendencje przejawia błąd rzeczywisty. Nie występuje zatem zjawisko przeuczenia.

Można zauważyć w przybliżeniu liniową zależność błędu klasyfikacji w funkcji poziomu zaszumienia, co świadczy o tym, że badane algorytmy w podobny sposób reagują na błędy podczas tworzenia zbioru trenującego.

Wykresy funkcji stability mają na celu zbadanie różnic klasyfikacji dwóch niezależnie zbudowanych klasyfikatorów. Wyniki świadczą o stabilności użytych algorytmów, gdyż wraz ze wzrostem liczby drzew w lesie coraz mniejszej ilości przykładów zostaje przydzielona różna kategoria.