1 Drzewa decyzyjne

Drzewa decyzyjne

- Reprezentują mechanizm decyzyjny.
- Węzły odpowiadają atrybutom.
- Krawędzie odpowiadają wyborom podejmowanym na podstawie atrybutów.
- Liście reprezentują finalne decyzje.
- W uczeniu maszynowym – konstruujemy drzewa na podstawie danych trenujących.

Indukcja drzew decyzyjnych — ID3

- Metoda indukcji drzew klasyfikacyjnych.
- Drzewa budowane są rekurencyjnie.
- Liście zawierają klasy — odpowiedzi modelu.
- Predykcja: dla danego x ustalana jest ścieżka w drzewie i zwracana klasa z liścia kończącego tą ścieżkę.

Indukcja drzew decyzyjnych — ID3

Algorithm 1: ID3

Input: Y: zbiór klas, D: zbiór atrybutów wejściowych, $U \neq \emptyset$: zbiór par uczących

if $\forall_{\{x_i,y_i\}\in U}\ y_i == y$ **then**
  return *Liść zawierający klasę* y
if $|D| == 0$ **then**
  return *Liść zawierający najczęstszą klasę w* U
$d = \arg\max_{d\in D} InfGain(d, U)$
$U_j = \{x_i, y_i\} \in U\ :\ x_i[d] = d_j$, gdzie d_j - j-ta wartość atrybutu d
return *Drzewo z korzeniem* d *oraz krawędziami* $d_j, j = 1, 2, \ldots$ *prowadzącymi do drzew: ID3(*Y*,* $D - \{d\}$*,* U_1*), ID3(*Y*,* $D - \{d\}$*,* U_2*),...*

Przykłady możliwych przypadków

Linie	Postać U			Węzeł
	x_1	x_2	y	
1–2	A	1	0	*Liść*(0)
	B	2	0	
	C	3	0	
			y	
3–4			0	*Liść*(1)
			1	
			1	
	x_1	x_2	y	
5–8	A	1	0	$Test(x_1)$
	B	2	1	
	B	3	1	

InfGain(d,U) — selekcja atrybutów do testu

- Entropia zbioru U:
$$I(U) = -\sum_i f_i \ln(f_i) \ ,$$
gdzie f_i - częstość i-tej klasy,
- Entropia zbioru podzielonego na podzbiory przez atrybut d:
$$Inf(d,U) = \sum_j \frac{|U_j|}{|U|} I(U_j)$$
- Zdobycz informacyjna:
$$InfGain(d,U) = I(U) - Inf(d,U)$$
.

Przykład: budowa drzewa

Zbudujemy drzewo za pomocą algorytmu ID3 dla następującego zbioru danych U:

x_1	x_2	y
A	1	0
B	1	1
B	2	1
B	2	0
B	3	1

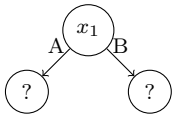
$$InfGain(x_1,U) = I(U) - Inf(x_1,U) \tag{1}$$
$$= I(U) - \sum_{j\in\{x_1=A, x_1=B\}} \frac{|U_j|}{|U|} I(U_j) \tag{2}$$
$$= 0.22 \tag{3}$$
$$InfGain(x_2,U) < InfGain(x_1,U) \tag{4}$$

Przyład: budowa drzewa

Krok 1 — U:

	x_1	x_2	y
1	A	1	0
2	B	1	1
3	B	2	1
4	B	2	0
5	B	3	1

Drzewo:

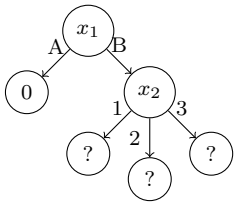


Krok 2 — U:

	x_2	y
1	1	0

	x_2	y
2	1	1
3	2	1
4	2	0
5	3	1

Drzewo:



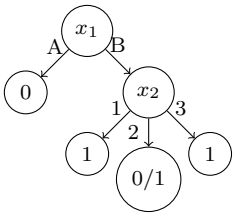
Krok 3 — U:

	y
2	1

	y
3	1
4	0

	y
5	1

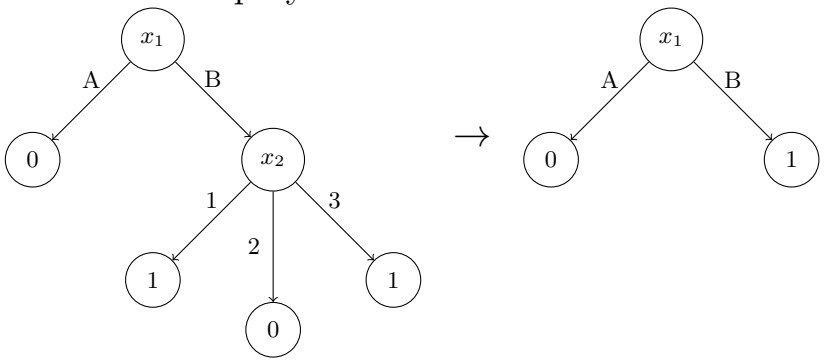
Drzewo:



Indukcja drzew decyzyjnych — algorytm C4.5

Problemem algorytmu ID3 jest konstruowanie drzew zbytnio rozbudowanych, i nadmiernie dopasowanych do zbioru uczącego. Rozwiązanie — metoda C4.5.

$C4.5 \approx ID3 + \text{przycinanie drzewa}$



Indukcja drzew decyzyjnych — główna idea $C4.5$

Algorithm 2: C4.5

Input: Y: zbiór klas, D: zbiór atrybutów wejściowych, $U \neq \emptyset$: zbiór par uczących

```
T = ID3(Y, D, U)
forall k ∈ T do Dla każdego liścia
    forall w ∈ path(s, root; T) do Dla każdego węzła na ścieżce liść-korzeń
        e_0 = oszacowanie błędu testowego w poddrzewie w
        e_1 = oszacowanie błędu testowego gdyby zmienić poddrzewo w na liść i zwracać najczęstszą
          klasę
        if e_0 ≥ e_1 then
            zastąp poddrzewo w liściem z najczęściej występującą w nim klasą
```

Szacowanie błędu testowego

Algorytm C4.5 wymaga wyliczania oszacowań błędu popełnianego przez poddrzewo na nowych danych. W praktyce, realizowane jest to przez zastosowanie odrębnego zbioru do przycinania lub z wykorzystaniem tzw. Obliczeniowej Teorii Uczenia się (COLT – *Computational Learning Theory*).

C4.5 — pominięte cechy

- Obsługa atrybutów ciągłych przez progowanie.
- Działanie w przypadku brakujących danych (nie uwzględniane w wyliczaniu entropii).
- Możliwość ważenia atrybutów.

ID3 vs C4.5

Oba podejścia nie są idealne.

- Wadą ID3 jest przeuczenie (*overfitting*)
- Wadą C4.5 zachłanny sposób wyboru atrybutów do węzłów drzewa, który ogranicza jakość klasyfikacji.

Pomysł:

- Zbudować wiele różnych drzew na bazie różnych zadań niesprzecznych z danym.
- Klasyfikacja — dominanta klasyfikacji dokonywanych przez drzewa.

Las losowy (*random forest*)

- Model składa się z wielu drzew zbudowanych niezależnie od siebie.
- Każde z drzew budowane jest na podzbiorze dostępnych danych.
- Predykcja przez agregację wyników.
- Model bardziej odporny na przeuczenie.

Las losowy (*random forest*) — trenowanie

Algorithm 3: TrainRandomForest

Input: Y: zbiór klas, D: zbiór atrybutów wejściowych, $U \neq \emptyset$: zbiór par uczących

forall $b = 1, \ldots, B$ **do**
  $U_b = B$ elementów wylosowanych z U ze zwracaniem
  $D_b = \lfloor \sqrt{|D|} \rfloor$ atrybutów wylosowanych z D bez zwracania
  $f_b =$ drzewo decyzyjne zbudowane na podstawie Y, U_b, D_b
end
return $F = \{f_1, \ldots, f_B\}$

Las losowy (*random forest*) — predykcja

Algorithm 4: PredictRandomForest

Input: x: wektor wejściowy, F: nauczony las losowy

$C = \emptyset$
forall $f_b \in F$ **do**
  $C = C \cup \{f_b(x)\}$
end
return *najczęstsza klasa ze zbioru* C

2 Gradient Boosting

Gradient boosting — idea

- Mamy zbiór uczący par postaci $\langle x_i, y_i \rangle \in \mathbb{R}^n \times \mathbb{R}^m$.
- Dla i-tego elementu, oznaczamy funkcję straty $q : \mathbb{R}^m \to \mathbb{R}$ jako q_i, np. $q_i(y) = ||y - y_i||^2$.
- Poszukujemy funkcji $f : \mathbb{R}^n \to \mathbb{R}^m$ minimalizującej stratę: $\sum_{i=1}^{N} q_i(f(x_i))$.
- Iteracyjnie konstruuje model złożony.
- Może służyć do rozwiązywania zadań regresji i klasyfikacji.
- W literaturze statystycznej, metoda występuje również pod nazwami MART (*Multiple Additive Regression Trees*) i GTBA (*Gradient Tree Boosting Algorithm*).

Gradient boosting — idea

- Postać modelu:

$$\hat{f}_k(x) = \hat{f}_0(x) + \sum_{i=1}^{k} \gamma_i \tilde{f}_i(x) \ ,$$

  gdzie $\gamma_i \in \mathbb{R}$, $\tilde{f}_i : \mathbb{R}^n \to \mathbb{R}^m$ to pojedyncze składniki.

- Funkcja konstruowana jest iteracyjnie — każda kolejna składowa $\tilde{f}_{k+1}$ poprawia jakość dotychczasowego modelu $\hat{f}_k$.
- Dodawanie składowej $\tilde{f} \rightarrow$ krok w kierunku zmniejszenia błędu całego modelu.

Gradient boosting — idea

- Postać modelu:

$$\hat{f}_k(x) = \hat{f}_0(x) + \sum_{i=1}^{k} \gamma_i \tilde{f}_i(x) \ .$$

- Analogia — zapiszmy metodę gradientu prostego przyjmując $\beta_k = -\alpha_k$:

$$\begin{aligned} \theta_{k+1} &= \theta_k - \alpha_k \nabla J(\theta_k) && (5)\\ &= \theta_k + \beta_k \nabla J(\theta_k) && (6)\\ &= \theta_0 + \sum_{i=0}^{k} \beta_i \nabla J(\theta_i) && (7) \end{aligned}$$

- W gradient boostingu „rolę gradientu" pełnią tzw pseudo residua:

$$r_{i,j} = -\left[\frac{\partial q_i\Big(\hat{f}_{j-1}(x_i)\Big)}{\partial \hat{f}_{j-1}(x_i)} \right] \ , \ i = 1, \ldots, N$$

Gradient boosting — algorytm

Algorithm 5: GradientBoosting

Input: U: zbiór uczący, q: funkcja straty

$\hat{f}_0 = \arg\min_y \sum_i q_i(y)$ ▷ `Tworzymy model stały`

forall $j = 1, \ldots, k$ **do**

  $r_{i,j} = -\left[\frac{\partial q_i\Big(\hat{f}_{j-1}(x_i)\Big)}{\partial \hat{f}_{j-1}(x_i)} \right] \ , \ i = 1, \ldots, N$ ▷ `Obliczamy pseudo residua - np.`

  $r_{i,j} = 2(y_i - \hat{f}_{j-1}(x_i))$ `dla kwadratowej funkcji straty`

  $\tilde{f}_j \leftarrow$ model wytrenowany na zbiorze uczącym $U_j = \big\{\langle x_i, r_{i,j}\rangle : i = 1, \ldots, N\big\}$

  $\gamma_j \leftarrow \arg\min_\gamma \sum_{i=1}^{N} q_i\big(\hat{f}_{j-1}(x_i) + \gamma \tilde{f}_j(x_i)\big)$

  $\hat{f}_j(x) = \hat{f}_{j-1}(x) + \gamma_j \tilde{f}_j(x)$

end

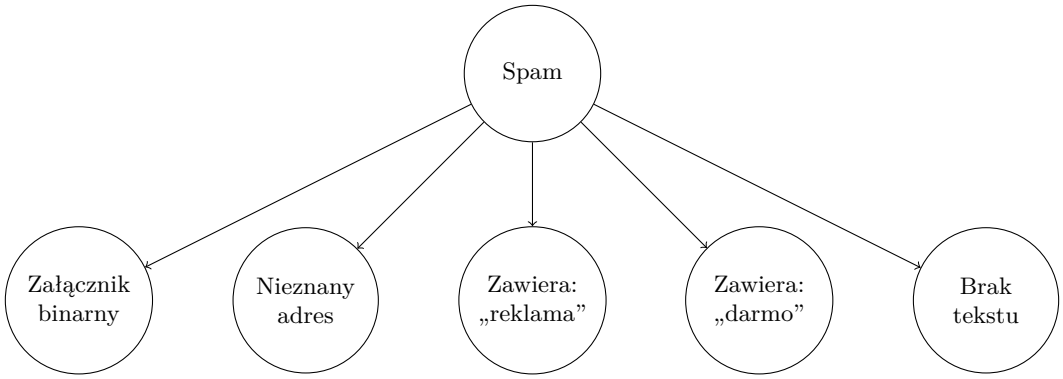
return f_k

XGBoost — eXtreme Gradient Boosting

- Implementacja metody Gradient Boosting.
- $\tilde{f}_j$ mają postać drzew.
- Do ściągnięcia z github-a.
- Projekt rozpoczęty przez Tianqi Chen'a z Distributed Machine Learning Community
- Często wygrywa konkursy na Kaggle.com.
- Kilkanaście hiperparametrów, które czasami trudno nastroić — domyślne wartości działają zazwyczaj całkiem dobrze.
- Wydajna implementacja, jest w stanie wykorzystać wszystkie rdzenie CPU.

Naiwny klasyfikator bayesowski

Przykład: wykrywanie spamu.



Twierdzenie Bayesa — zastosowanie w klasyfikacji

Zakładając, że y to przewidywana klasa, a $x_1, \ldots, x_n$ - atrybuty wejściowe i stosując tw. Bayesa, otrzymujemy:

$$\begin{aligned} P(y|x_1, \ldots, x_n) &= \frac{P(y, x_1, \ldots, x_n)}{P(x_1, \ldots, x_n)} \\ &= \frac{P(x_1, \ldots, x_n|y)P(y)}{P(x_1, \ldots, x_n)} \end{aligned}$$

Naiwny klasyfikator bayesowski

- Do zbudowania modelu na podstawie tw. Bayesa potrzebujemy estymat:

$$P(x_1, \ldots, x_n|y)$$

- Jeśli („naiwnie”) założymy, że atrybuty wejściowe są wzajemnie niezależne:

$$P(x_1, \ldots, x_n|y) = \prod_{i=1}^{n} P(x_i|y)$$

- Klasę dla danego $\mathbf{x} = [x_1, \ldots, x^n]^T$ wyliczamy:

$$\hat{y} = \arg\max_{y \in Y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

- Wartości $P(u)$ oraz $\forall_{i=1,\ldots,n} P(x_i|y)$ estymujemy na podstawie danych uczących.

3 Budowanie modeli UM

Budowa modelu UM — typowe kroki

1. Zdefiniowanie zadań modelowania.
2. Zebranie i analiza danych.
3. Feature engineering, budowanie modeli, strojenie hiperparametrów, ...
4. Wybór i przygotowanie finalnego modelu —- jeśli nie jest dostatecznie dobry, to idź to pkt. 2.

Analiza danych — na co zwracać uwagę?

- Rozkłady atrybutów wejściowych.
- Brakujące dane.
- Błędne wartości.
- Anomalie/Elementy odstające.
- „Wycieki" atrybutów wyjściowych – np. prognoza pogody bez przesunięcia czasowego.
- W przypadku klasyfikacji: niesymetryczne rozkłady klas.
- W przypadku regresji: czy zmienna przewidywana ma liniowy czy wykładniczy charakter.

Jakość modelu — kilka pojęć

Generalizacja — jakość/dokładność modelu dla danych spoza zbioru uczącego.

Przeuczenie (*Overfitting*) — nadmierne dopasowanie do zbioru uczącego powodujące brak zdolności do generalizacji.

Niedotrenowanie *Underfitting* — niezdolność modelu do dopasowania do zbioru trenującego.

3.1 Przeuczenie

Przyczyny przeuczenia/braku generalizacji

1. Zbyt skomplikowana postać modelu.
2. Za długo trwająca optymalizacja parametrów modelu.
3. Za mały zbiór danych uczących.
4. Niereprezentatywny zbiór danych uczących.

Przeciwdziałanie przeuczeniu

1. Kontrolowanie struktury modelu —- bardziej skomplikowany nie zawsze znaczy lepszy.
2. Stosowanie agregacji — koncepcje pokrewne do lasów losowych.
3. Zastosowanie regularyzacji podczas trenowania.
4. Tzw. *dropout* w modelach neuronowych.

Typowe techniki regularyzacji

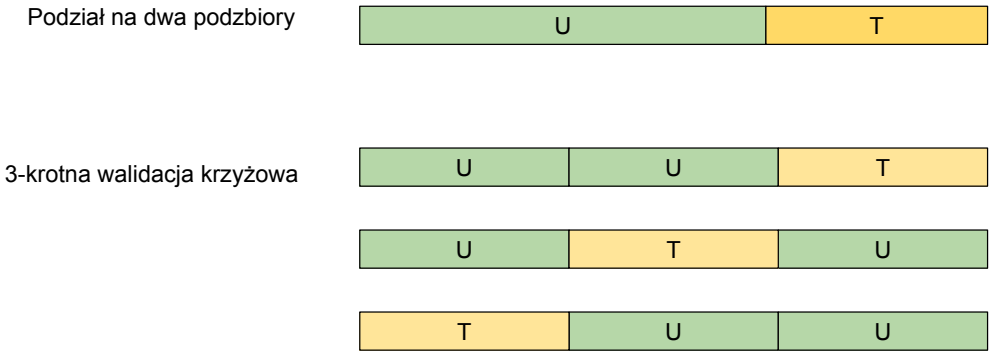
- Podejście sprowadza się do rozszerzenia funkcji straty o składową karzącą za zbytnio rozbudowaną strukturę modelu.
- Przykłady funkcji straty dla zadania regresji:
  - regularyzacja L_1: $J(\theta) = \frac{1}{N}\sum_{i=1}^{N} ||y_i - \hat{f}(x_i, \theta)||^2 \ + \ \lambda||\theta||$
  - regularyzacja L_2: $J(\theta) = \frac{1}{N}\sum_{i=1}^{N} ||y_i - \hat{f}(x_i, \theta)||^2 \ + \ \lambda||\theta||^2$

3.2 Ocena jakości modelu

Ocena jakości modelu

- Ze względu na groźbę przeuczenia, ocenę jakości modelu trzeba wykonywać na danych, które nie były stosowane do uczenia.
- W przypadku strojenia hiper-parametrów metody uczącej, sytuacja jest analogiczna.
- Najbardziej podstawowa strategia podział na zbiór uczący i testowy.
- "Typowe"proporcje 80:20, 75:25.

k-krotna walidacja krzyżowa



k-krotna walidacja krzyżowa

Algorithm 6: WalidacjaKrzyżowa

Input: $U \neq \emptyset$: zbiór par uczących, k - liczba podziałów

1 Sortujemy zbiór U w losowej kolejności.

`/* Dzielimy` U `na` k `równych części */`

2 $U_1 \cup \ldots \cup U_k = U$

3 **forall** $i = 1, \ldots, k$ **do**

4 $\quad \hat{f}_i \leftarrow$ Uczymy model na zbiorze $U - U_i$

5 $\quad e_i \leftarrow$ średnią strata modelu $\hat{f}_i$ na zbiorze U_i

6 **end**

`/* Wyliczamy średnią stratę */`

7 $e \leftarrow \frac{1}{k} \sum_{i=1}^{k} e_i$

8 $\hat{f} \leftarrow$ Uczymy model na zbiorze U

9 **return** $\hat{f}, e$

3.3 Podsumowanie

Klasyfikacja i regresja – podsumowanie

- Elementy: dane uczące, funkcja straty, optymalizator.
- Modele/Architektury: liniowe, wielomiany, jądrowe, drzewa, modele zagregowane, sieci neuronowe, itd.
- Zastosowania: rozpoznawanie twarzy, przewidywanie notowań giełdowych/wskaźników ekonometrycznych, rozpoznawanie pisma ręcznego, ocena jakości produktów (np. gatunków wina), przewidywanie parametrów urządzeń technicznych, itd.