Rozdział 2

Podstawowe pojęcia

2.1 Modele klasyfikujące

Uczenie maszynowe to zagadnienie, które w świecie nauki zostało po raz pierwszy wykorzystane relatywnie niedawno, bo w drugiej połowie XX wieku. Tego terminu po raz pierwszy użył Arthur Samuel w swojej publikacji [Sam59a] z roku 1959. W publikacji można znaleźć kluczowe dwa stwierdzenia w kontekście przybliżenia terminu uczenia maszynowego jako takiego, a mianowicie „zaprogramowanie komputerów do uczenia się na podstawie doświadczeń [Sam59b, tłumaczenie własne, s. 535] oraz „uczenie się poprzez generalizację (doświadczeń)" [Sam59b, tłumaczenie własne, s. 546]. Z kolei Tom Mitchell definiuje uczenie pisząc, że „program komputerowy uczy się rozwiązywania zadania T w oparciu o doświadczenie E i miarę jakości P, jeżeli wraz z przyrostem doświadczenia E poprawia się jakość wykonywania zadania T, którego wykonanie jest oceniane miarą P" [Mit97, tłumaczenie własne, s. 575-576].

W kontekście modeli klasyfikujących trzeba zdefiniować pojęcie klasyfikacji, modelu oraz jego hiperparametrów. Stanisław Osowski w swoim podręczniku definiuje pojęcia klasyfikacji i modelu w następujący sposób:

- klasyfikacja to „proces regresji, w którym na podstawie analizy i przetwarzania atrybutów wejściowych generowana jest etykieta klasy" [Oso13, s. 23]. Klasyfikacja jest klasyfikacją binarną, kiedy dane etykietowane są dwoma wartościami. Przykładem takiej klasyfikacji jest decydowanie czy dana wiadomość e-mail to spam czy nie na podstawie jej treści. Klasyfikacja jest klasyfikacją wieloklasową, kiedy dane są etykietowane więcej niż dwoma wartościami,
- model „odnosi się do sposobu przetwarzania danych (algorytmu) z kompletnym zestawem parametrów, dobranych zwykle w procesie uczenia. Jego postacią może być wzór matematyczny lub określona struktura systemu" [Oso13, s. 23].

Proces przekształcania danych, uczenia i klasyfikacji w myśl przytoczonych definicji nazywamy modelem klasyfikującym. Dodatkowo, hiperparametry to parametry algorytmów, które ustawiane są przed procesem uczenia i nie są one modyfikowane podczas procesu treningu. Przykładem hiperpara-

metru dla algorytmu lasu losowego jest liczba jego drzew.

Do procesu uczenia nadzorowanego niezbędne są oznaczone, usystematyzowane i zagregowane dane. Niech zbiór danych uczących jest oznaczony jako D_u. Zbiór danych uczących jest uzyskiwany poprzez dokonanie różnego rodzaju przekształceń na zbiorze danych wejściowych D_{wej}. Jeżeli nie istnieje z góry określony zbiór danych testowych D_t, który jest poddawany takim samym przekształceniom, co D_u, wtedy zbiór danych uczących będzie podzielony na zbiór danych treningowych i zbiór danych testowych. Czasem dokonuje się podziału zbioru uczącego na zbiór danych: testowych, treningowych i walidacyjnych. Podziału dokonuje się ze względu na potrzeby ewaluacji modelu. Ewaluacja modelu przy pomocy zbioru danych testowych, który identyczny względem zbioru danych treningowych jest błędną metodą pomiaru miary jakości modelu. Przykładowy zbiór danych wejściowych D_{wej} przedstawiono w formie tabeli 2.1.

cecha$_1$	**cecha**$_2$	**ceha**$_3$	**klasa**
Kobieta	Nihil fit sine causa	25.5	Nie kupi
Mężczyzna	Carpe diem	22.5	Kupi
Mężczyzna	Memento mori	40	Nie kupi
Kobieta	Nihil fit sine causa	26.5	Nie kupi
...	...	...	...

Tabela 2.1: Przedstawiony w formie tabelarycznej przykładowy zbiór danych wejściowych D_{wej}. Źródło: opracowanie własne

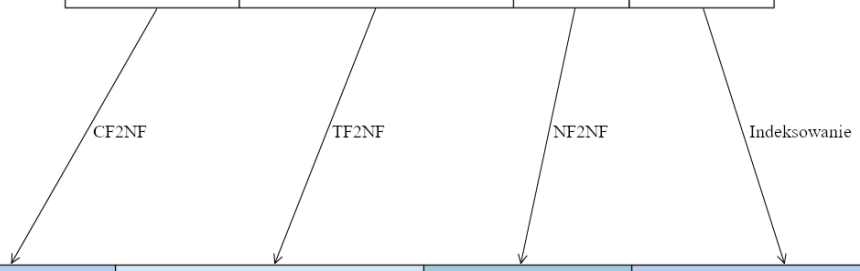
Na dane wejściowe składają się obserwacje. Obserwacja jest pojedynczym rekordem danych, który składa się z wartości cech numerycznych, kategorialnych lub tekstowych i wybranego atrybutu decyzyjnego (klasy, etykiety), który opisuje obserwację. Natomiast zbiór danych uczących jest zbiorem krotek zbudowanych z deskryptorów numerycznych, czyli liczbowej lub wektorowej reprezentacji danych wartości i zindeksowanego atrybutu decyzyjnego. Przykładowy zbiór danych uczących przedstawiono w formie tabeli 2.2.

Aby przekształcić cechę numeryczną, kategorialną lub tekstową w deskryptor numeryczny należy zastosować na niej odpowiednią transformację z danej klasy przekształceń. W pracy przyjmuje się, że cechy numeryczne są oznaczane jako NF, kategorialne jako CF, a tekstowe jako TF. Dla NF odpowiednim przekształceniem będzie $NF2NF$, dla CF – $CF2NF$, a dla TF – $TF2NF$. Przykładowe przekształcenie D_{wej} w D_u zostało zobrazowane na rysunku 2.1.

deskryptor$_1$	**deskryptor**$_2$	**deskryptor**$_3$	**zindeksowana klasa**
0	[0, 0.24, 0, 0, 0]	3	1
1	[0, 0.11, 0, 0, 0.11]	1	0
1	[0.52, 0.69, 0.32, 0, 0]	17.5	1
0	[0, 0.24, 0, 0, 0]	3.25	1
...	...	...	...

Tabela 2.2: Przedstawiony w formie tabelarycznej przykładowy zbiór danych uczących D_u. Źródło: opracowanie własne

cecha$_1$	**cecha**$_2$	**ceha**$_3$	**klasa**
Kobieta	Nihil fit sine causa	25.5	Nie kupi
Mężczyzna	Carpe diem	22.5	Kupi
Mężczyzna	Memento mori	40	Nie kupi
Kobieta	Nihil fit sine causa	26.5	Nie kupi
...	...	...	...

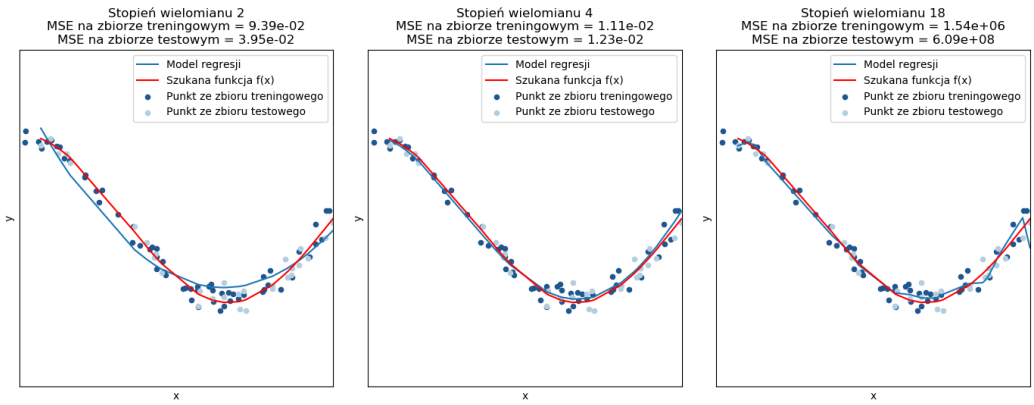


deskryptor$_1$	**deskryptor**$_2$	**deskryptor**$_3$	**zindeksowana klasa**
0	[0, 0.24, 0, 0, 0]	3	1
1	[0, 0.11, 0, 0, 0.11]	1	0
1	[0.52, 0.69, 0.32, 0, 0]	17.5	1
0	[0, 0.24, 0, 0, 0]	3.25	1
...	...	...	...

Rysunek 2.1: Przekształcenie D_{wej} w D_u przy pomocy odpowiednich klas transformacji. Cechy zostały przekształcone w deskryptory numeryczne za pomocą przykładowego przekształcenia z klasy przekształceń $CF2NF$, $TF2NF$ oraz $NF2NF$, a wartości klasy zostały zindeksowane. Źródło: opracowanie własne

Celem poprawnej klasyfikacji jest znalezienie takiej funkcji c, która dla każdego wektora x złożonego z deskryptorów numerycznych w oparciu o uogólnioną wiedzę odkrytą podczas procesu uczenia (treningu), przypisze odpowiednią wartość klasy y. Uogólniony klasyfikator to taki, który nie jest przetrenowany.

Zjawisko przetrenowania lub nadmiernego dopasowania (ang. overfitting) modelu najłatwiej można zaobserwować na przykładzie z rysunku 2.2 z przedstawionym problemem budowy modelu regresji wielomianowej. Szukaną funkcją jest funkcja $f(x) = cos(\frac{3}{2}\pi x)$. Zbiór treningowy zawiera 60 punktów, a zbiór testowy 30 punktów losowo wygenerowanych w otoczeniu szukanej funkcji. Przecięcie tych dwóch zbiorów jest zbiorem pustym. Na kolejnych wykresach zostały przedstawione trzy modele: wielomianowy stopnia drugiego, wielomianowy stopnia czwartego oraz wielomianowy stopnia osiemnastego i ich wartości błędu średniokwadratowowego uzyskane na zbiorze treningowym oraz na zbiorze testowym. Błąd średniokwadratowy definiuje się jako $\sqrt{\frac{1}{n}\sum_{i=1}^{n}|d_i - y_i|^2}$, gdzie y to wartość estymowana przez model, d to wartość dokładna, a n oznacza całkowitą liczbę obserwacji [Oso13, s. 200-201]. Wartość bezwzględna różnic wartości błędów średniokwadratowych pomiędzy miarą uzyskaną na zbiorze treningowym i na zbiorze testowym jest najmniejsza dla modelu z wielomianem stopnia czwartego $(12 * 10^{-2})$, a największa dla modelu z wielomianem stopnia osiemnastego $(607,46 * 10^{6})$.



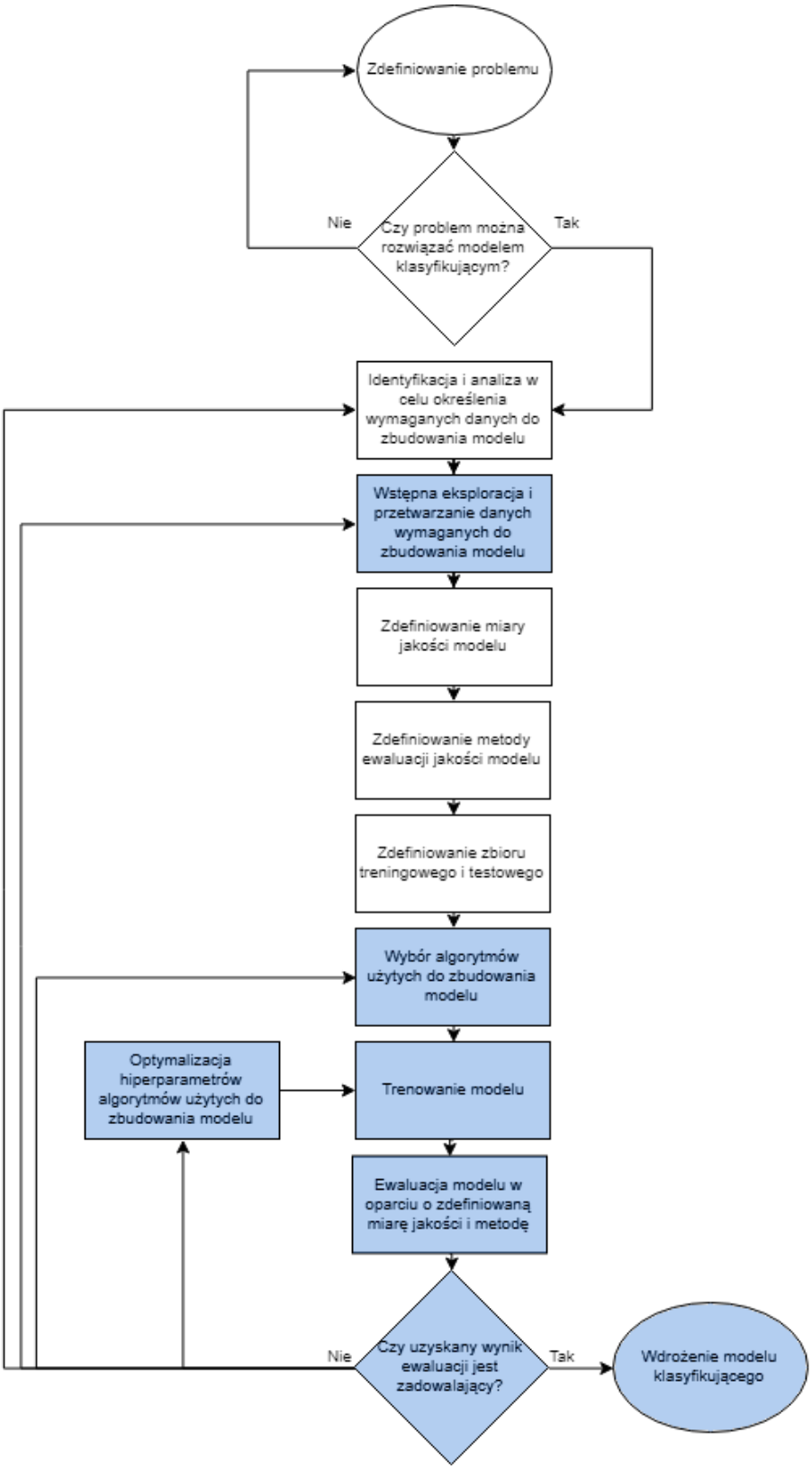
Rysunek 2.2: Przykładowy problem regresji. Źródło: opracowanie własne na podstawie https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

Na podstawie otrzymanych wyników można stwierdzić, że model wielomianowy stopnia czwartego najlepiej przybliża szukaną funkcję. Natomiast w przypadku wielomianu stopnia osiemnastego pomimo, że ten przeszedł przez niektóre z punktów ze zbioru treningowego, to wskazana wyżej wartość bezwzględna różnicy błędów jest bardzo duża ze względu na wysoką miarę błędu średniokwadratowego uzyskanego na zbiorze testowym. Ze względu na najniższą różnicę uzyskaną dla modelu wielomianowego stopnia czwartego można stwierdzić, że utworzony model w procesie uczenia odkrył wiedzę zawartą w danych treningowych i cechuje się uogólnieniem.

Zatem na podstawie powyższego przykładu zakłada się, że model klasyfikujący nie będzie cechować się przeuczeniem, jeśli wartość bezwzględna różnicy wartości miary jakości klasyfikatora uzyskana na zbiorze treningowym i testowym lub dowolnie innym zbiorze, którego część wspólna ze zbiorem treningowym jest pusta, jest odpowiednio niska. Algorytm wyboru nieprzeuczonych modeli na podstawie wartości ich miar jakości z wykorzystaniem k-krotnej walidacji krzyżowej, uzyskanych na zbiorze treningowym, testowym i walidacyjnym jest przedstawiony w podrozdziale 2.1.3.1.

2.1.1 Budowa modelu

Sam wybór algorytmów, cech czy odpowiednich przekształceń do zbudowania modelu klasyfikującego mimo tego, że jest kluczowy, jest niewystarczający. Warunkiem koniecznym do budowy modelu są odpowiednie rozłączne dane treningowe i testowe. Niemniej na początku zawsze należy zastanowić się czy model klasyfikujący to odpowiedni sposób do rozwiązania danego problemu. Jeżeli na tym etapie dane nie są znane, to należy zastanowić jakie powinny być wymagane i jakie najlepiej mogą opisywać predykowane klasy. Zatem proponowane rozwiązanie powinno zostać dopasowane adekwatnie do rozwiązywanego problemu. Czasem może okazać się, że przykładowo, przy dużej liczbie wartości atrybutu deskryptywnego będącego liczbą całkowitą, lepszym rozwiązaniem rozważanego problemu ze względu na jego naturę, będzie model regrsji. Dodatkowo bardzo ważnym problemem jest sposób weryfikacji modelu i wybór metody jego ewaluacji. Kolejne kroki budowy modelu klasyfikującego są przedstawione na diagramie 2.3. Kolorem niebieskim zaznaczono kroki, które mogą zostać zautomatyzowane w ramach zautomatyzowanego uczenia maszynowego.



Rysunek 2.3: Proces budowy modelu klasyfikującego. Kolorem niebieskim zaznaczono kroki, które mogą zostać zautomatyzowane. Źródło: opracowanie własne na podstawie [Kot07] [DB15]

2.1.2 Miary jakości

Miary jakości dla klasyfikacji binarnej i wieloklasowej, sposób ich wyliczenia oraz przedmiot ich oceny przedstawiono odpowiednio w tabeli 2.4 oraz w tabeli 2.5. Składowymi wzorów są składowe tablicy pomyłek. Wartości prawdziwie pozytywnych, fałszywie pozytywnych, fałszywie negatywnych i prawdziwie negatywnych predykcji klasyfikatora są wyliczane dla każdej predykowanej klasy z osobna.

		Klasa rzeczywista	
		Pozytywna	Negatywna
Klasa predykowana	Pozytywna	prawdziwie pozytywna (TP)	fałszywie pozytywna (FP)
	Negatywna	fałszywie negatywna (FN)	prawdziwie negatywna (TN)

Tabela 2.3: Tablica pomyłek (ang. confusion matrix). Źródło: opracowanie własne na podstawie [SL09]

W tabelach 2.4 i 2.5 β to skalar taki, że $\beta \geqslant 1$.

Miara	Wzór	Przedmiot oceny
Skuteczność (ang. accuracy)	$\frac{TP+TN}{TP+FN+FP+TN}$	Ogólna skuteczność klasyfikatora.
Stopa błędu (ang. error rate)	$\frac{FP+FN}{TP+TN+FP+FN}$	Średni błąd względny.
Precyzja (ang. precision)	$\frac{TP}{TP+FP}$	Jaka część przypadków zaklasyfikowanych jako TP jest w rzeczywistości TP.
Czułość (ang. recall)	$\frac{TP}{TP+FN}$	Skuteczność klasyfikatora w poprawnym identyfikowaniu TP.
Miara F (ang. F-score)	$\frac{(\beta^2+1)TP}{(\beta^2+1)TP+\beta^2FN+FP}$	Balans pomiędzy czułością, a precyzją.
Specyficzność (ang. specificity)	$\frac{TN}{FP+TN}$	Skuteczność klasyfikatora w poprawnym identyfikowaniu TN.
AUC	$\frac{1}{2}(\frac{TP}{TP+FN}+\frac{TN}{TN+FP})$	Zdolność klasyfikatora do unikania fałszywej klasyfikacji.

Tabela 2.4: Miary jakości klasyfikacji binarnej. Źródło: opracowanie własne na podstawie [SL09]

Miara	Wzór
Średnia skuteczność (ang. average accuracy)	$\frac{\sum_{i=1}^{C} \frac{TP_C+TN_C}{TP_C+FN_C+FP_C+TN_C}}{C}$
Średnia stopa błędu (ang. average error rate)	$\frac{\sum_{i=1}^{C} \frac{FP_C+FN_C}{TP_C+FN_C+FP_C+TN_C}}{C}$
Precyzja mikro (ang. precision_μ)	$\frac{\sum_{i=1}^{C} TP_C}{\sum_{i=1}^{C}(TP_C+FP_C)}$
Precyzja makro (ang. precision_M)	$\frac{\sum_{i=1}^{C} \frac{TP_C}{TP_C+FP_C}}{C}$
Czułość mikro (ang. recall_μ)	$\frac{\sum_{i=1}^{C} TP_C}{\sum_{i=1}^{C}(TP_C+FN_C)}$
Czułość makro (ang. recall_M)	$\frac{\sum_{i=1}^{C} \frac{TP_C}{TP_C+FN_C}}{C}$
Miara F mikro (ang. Fscore_μ)	$\frac{(\beta^2+1)Precision_\mu Recall_\mu}{\beta^2 Precision_\mu+Recall_\mu}$
Miara F makro (ang. Fscore_M)	$\frac{(\beta^2+1)Precision_M Recall_M}{\beta^2 Precision_M+Recall_M}$

Tabela 2.5: Miary jakości klasyfikacji wieloklasowej. Źródło: opracowanie własne na podstawie [SL09]

	Przykładowy klasyfikator a	Przykładowy klasyfikator B
Klasa rzeczywista	Klasa predykowana	
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	2	2
1	2	2
2	1	1
2	1	1
2	2	2
2	3	3
2	3	3
3	3	1
3	3	1

Tabela 2.6: Przykładowe wyniki predykcji dwóch przykładowego klasyfikatora a i przykładowego klasyfikatora B. Źródło: opracowanie własne

Najczęściej wykorzystywaną miarą klasyfikacji jest skuteczność „określająca procent dobrze sklasyfikowanych przypadków" [Oso13, s.206]. Miara ta jest miarą ogólną. Nie powinna ona być jednak wykorzystywana „z wyboru", ponieważ ten powinien wynikać z dokonanych założeń, co do celu klasyfikacji. Wybór ważny jest tym bardziej, jeżeli dla zbioru treningowego i testowego spodziewany jest

problem niezrównoważenia klas, szczególnie w klasyfikacji wieloklasowej, a celem budowanego modelu jest poprawna klasyfikacja obserwacji etykietowanych rzadko występującym atrybutem deskryptywnym. Przykład takiego problemu został przedstawiony w tabelach 2.6, 2.7 oraz 2.8. O problemie niezrównoważenia klas mówi się wtedy, kiedy liczba obserwacji dla każdego atrybutu deskryptywnego nie jest w przybliżeniu równoliczna (rozkład klas nie jest w przybliżeniu jednorodny).

Miara	Klasyfikator	Klasa	Wartość
TP	A	1	7
		2	1
		3	2
	B	1	7
		2	1
		3	0
FP	A	1	2
		2	2
		3	2
	B	1	4
		2	2
		3	2
FN	A	1	2
		2	4
		3	0
	B	1	2
		2	4
		3	2
TN	A	1	5
		2	9
		3	12
	B	1	3
		2	9
		3	12

Tabela 2.7: Wartości składowych tablicy pomyłek dla przykładowego klasyfikatora a oraz przykładowego klasyfikatora B i poszczególnych klas na podstawie wyników z tabeli 2.6. Źródło: opracowanie własne

Aby zobrazować problem wyboru odpowiedniej miary, w tabeli 2.6 przedstawione zostały wyniki predykcji dwóch przykładowych klasyfikatorów dla szesnastu obserwacji. Klasę 1 opisuje dziewięć obserwacji, klasę 2 – pięć obserwacji, klasę 3 - dwie obserwacje. W tabeli 2.7 przedstawiono wartości składowych tablicy pomyłek dla poszczególnych klas. Wyniki predykcji klasyfikatora A różnią się od wyników klasyfikatora B jedynie klasą predykowaną dla obserwacji rzadkiej klasy 3. Klasyfikator A wskazał poprawnie predykcję dla obserwacji z tą klasą, a klasyfikator B nie. Dziesięć na szesnaście

predykcji klasyfikatora A i osiem na szesnaście predykcji klasyfikatora B jest poprawnych. Analizując zestawienie przedstawione w tabeli 2.8 można zauważyć, że miara średniej skuteczności dla klasyfikatora A jest akceptowalna, a dla B jest niezbyt zadowalająca (średnia skuteczność dla klasyfikatora A to 0.75, a dla klasyfikatora B to $0,6666$). Natomiast wartości czułości i precyzji w skali makro są niskie (czułość makro klasyfikatora A to $0,6592$, a klasyfikatora B to $0,5370$, precyzja makro dla klasyfikatora A to $0,3259$, a dla klasyfikatora B to $0,3232$), a ich różnice są znacznie większe niż różnica średniej skuteczności. Powyższy przykład ilustruje, że jeżeli założonym celem klasyfikacji jest równie skuteczna predykcja każdej klasy z osobna lub poprawna predykcja klasy rzadkiej, to wybór miary średniej skuteczności nie jest w tym przypadku uzasadniony.

Miara		Klasyfikator	Wartość
Mikro	Czułość	A	0,625
		B	0,5
	Prezycja	A	0,625
		B	0,5
	Miara F	A	0,625
		B	0,5
Makro	Czułość	A	0,6592
		B	0,3259
	Precyzja	A	0,5370
		B	0,3232
	Miara F	A	0,5919
		B	0,3245
Średnia skuteczność		A	0,75
		B	0,6666
Średnia stopa błędu		A	0,25
		B	0,3333

Tabela 2.8: Zestawienie wartości miar dla klasyfikatora a oraz B na podstawie wyników z tabeli 2.6. Źródło: opracowanie własne

Ustalenie odpowiedniej miary jest bardzo ważne z tego względu, że w realnych przypadkach użycia wraz z upływem czasu będą pojawiać się nowe rekordy danych. Być może kolejne obserwacje będą zawierać wcześniej nieznany atrybut decyzyjny, a w kolejnej iteracji trenowania modelu takich obserwacji będzie stosunkowo niewiele zarówno w zbiorze treningowym jak i testowym. Dodatkowo warto pamiętać, że „maksymalizacja jednego wskaźnika jest często sprzężona z pogorszeniem innego" [Oso13, s.206], dlatego dobór miary do konkretnego problemu jest niezwykle istotny.

2.1.3 Sposoby ewaluacji

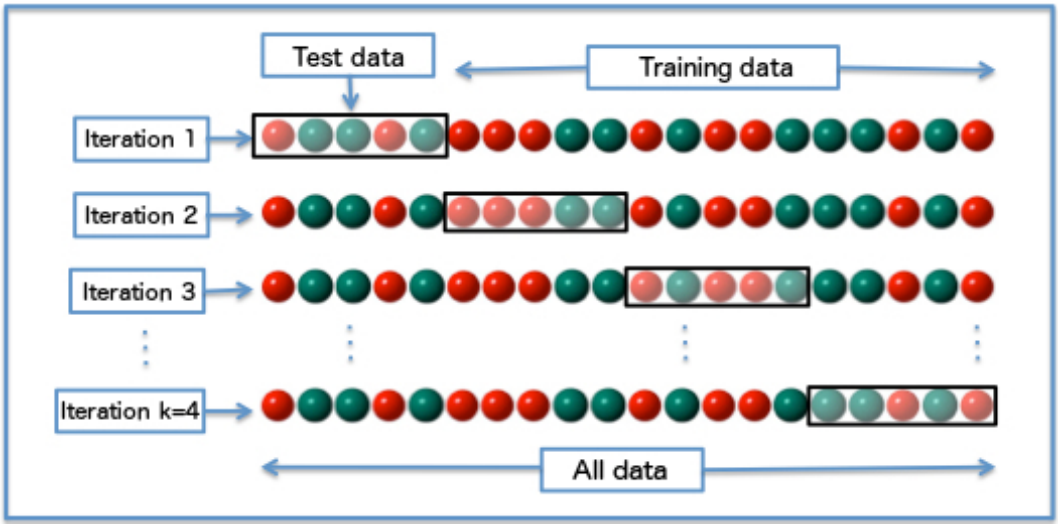
Aby uzyskać wyniki miary jakości modelu potrzeba sposobu na jego ewaluację. Ewaluacja powinna zostać wykonana na zbiorze testowym, który jest zbiorem rozłącznym ze zbiorem treningowym. Roz-

łączność tych zbiorów jest kluczowa, ponieważ w trakcie uczenia dobierane są optymalne parametry algorytmów na bazie danych – na przykład podział drzewa w algorytmie drzewa decyzyjnego. Jeżeli zbiór danych testowych jest równocześnie zbiorem danych treningowych, to istnieje wysokie prawdopodobieństwo, że uzyskane metryki w ramach ewaluacji są bezużyteczne, ponieważ skutkiem takiego uczenia jest nadmierne dopasowanie modelu do danych zawartych w zbiorze treningowym, a nie jego generalizacja. Rozkład klas w zbiorze danych testowych i zbiorze danych treningowych powinien być porównywalny.

Istnieje kilka sposobów ewaluacji wybranej miary jakości i sztucznego podziału zbioru treningowego na treningowy i testowy. Jednym z nich jest podział danych uczących „w proporcji zbliżonej do 2:1 lub 3:1, w której większa część służy do uczenia, a pozostała do testowania. Podział danych jest najczęściej losowy. Wiąże się z tym problem losowości uzyskanych wyników testowania. Jednokrotny test może nie być reprezentatywny i przyjęcie jego wyników jako obiektywnych nie jest uzasadnione" [Oso13, s. 213]. Dlatego taki test i podział należy przeprowadzić wielokrotnie uśredniając jego wyniki albo wskazując medianę jego wyników. Czasem zdarza się, że podziału zbioru danych uczących dokonuje się ręcznie z zachowaniem proporcji klas z równoczesnym ich równoważeniem, o ile to możliwe.

2.1.3.1 K-krotna walidacja krzyżowa

Inną popularną metodą podziału danych treningowych i ewaluacji modelu na zbiorze danych testowych jest k-krotna walidacja krzyżowa. Polega ona na losowym i o ile to możliwe, równym podziale zbioru danych treningowych na k części „najczęściej na 10" [Oso13]. Wtedy $k-1$ części zostaje użyte jako nowy zbiór treningowy będący podzbiorem zbioru treningowego. Pozostała część to zbiór walidacyjny będący podzbiorem zbioru treningowego. Zbiór walidacyjny służy do testowania modelu uzyskanego na nowym zbiorze treningowym. Przykładowy podział został zobrazowany na rysunku 2.4.



Rysunek 2.4: Zobrazowanie podziału zbioru danych treningowych w ramach 4-krotnej walidacji . Źródło: https://en.wikipedia.org/wiki/Cross-validation_(statistics), rysunek autorstwa Fabian Flöck

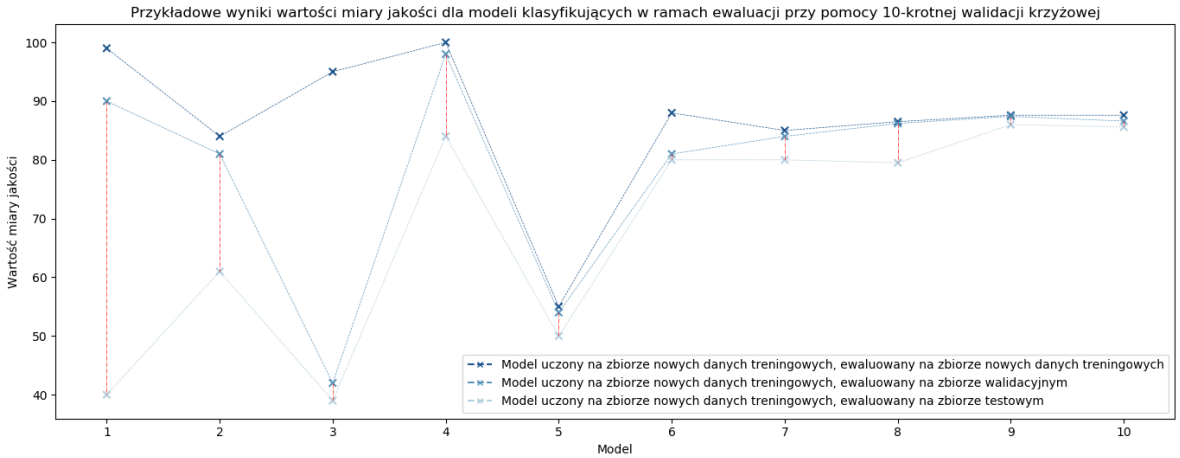
Gdy w kolejnych podziałach na nowy zbiór treningowy i zbiór walidacyjny zachowany zostaje rozkład klas, to wtedy taka walidacja jest k-krotną krzyżową walidacją stratyfikowaną. W wyniku k-krotnej walidacji krzyżowej powstaje k modeli. Zazwyczaj spośród uzyskanych modeli wybierany i testowany na zbiorze danych testowych jest ten, który spośród k modeli uzyskał najwyższą wartość miary jakości modelu. Wartość miary uzyskana na zbiorze danych testowych dla takiego modelu jest ostatecznym wynikiem ewaluacji modelu. W ramach pracy, aby zminimalizować ryzyko wskazania modelu, który mógłby być modelem przeuczonym, przyjmuje się, że wskazanym najlepszym modelem spośród k modeli jest ten wybrany według następującego algorytmu:

1. Dla każdego k-tego modelu nauczonego na nowym zbiorze treningowym:
   (a) Uzyskaj wartość miary jego jakości poprzez testowanie modelu na zbiorach:
      - nowym treningowym – wartość jego miary jest oznaczona jako ntr_k,
      - walidacyjnym – wartość jego miary jest oznaczona jako w_k,
      - testowym – wartość jego miary jest oznaczona jako t_k

   (b) Oblicz:
      - $d_1 = |ntr_k - t_k|$,
      - $d_2 = |ntr_k - w_k|$,
      - $d_3 = |w_k - t_k|$

   (c) Wtedy $t = \frac{d_1}{d_2+1} * 2$.
   (d) Jeżeli $d_3 \leqslant t$, to uznaj, że dany model jest nieprzeuczony i dodaj go do listy wyników wraz z wartościami w_k i t_k.
2. Jeśli lista wyników jest niepusta, to uporządkuj listę wyników sortując je rosnąco względem t_k, a następnie malejąco względem w_k. Wskaż pierwszy model z listy jako ten najlepszy.
3. Jeśli lista wyników jest pusta, to nie zostanie wskazany żaden model.

Wyniki działania powyższego algorytmu dla przykładowych wyników 10-krotnej walidacji krzyżowej z rysunku 2.5 zostały zaprezentowane w tabelach 2.9 i 2.10.



Rysunek 2.5: Przykładowe wyniki 10-krotnej walidacji krzyżowej. Źródło: opracowanie własne

k-ty model	Wartość miary jakości uzyskana na zbiorze			d_1	d_2	d_3	t
	Nowym treningowym	Walidacyjnym	Testowym				
1	99,00	90,00	40,00	59,00	9,00	50,00	11,80
2	84,00	81,00	61,00	23,00	3,00	20,00	11,50
3	95,00	42,00	39,00	56,00	53,00	3,00	2,07
4	100,00	98,00	84,00	16,00	2,00	14,00	10,67
5	55,00	54,00	50,00	5,00	1,00	4,00	5,00
6	88,00	81,00	80,00	8,00	7,00	1,00	2,00
7	85,00	84,00	80,00	5,00	1,00	4,00	5,00
8	86,50	86,20	79,50	7,00	0,30	6,70	10,77
9	87,60	87,40	86,00	1,60	0,20	1,40	2,67
10	87,60	86,60	85,60	2,00	1,00	1,00	2,00

Tabela 2.9: Wyniki zawierające modele uważane za przeuczone i nieprzeuczone uzyskane przy pomocy algorytmu 2.1.3.1 dla wyników przedstawionych na rysunku 2.5. Źródło: opracowanie własne

k-ty model	Wartość miary jakości uzyskana na zbiorze			d_1	d_2	d_3	t
	Nowym treningowym	Walidacyjnym	Testowym				
10	**87,60**	**86,60**	**85,60**	**2,00**	**1,00**	**1,00**	**2,00**
6	88,00	81,00	80,00	8,00	7,00	1,00	2,00
9	87,60	87,40	86,00	1,60	0,20	1,40	2,67
7	85,00	84,00	80,00	5,00	1,00	4,00	5,00
5	55,00	54,00	50,00	5,00	1,00	4,00	5,00
8	86,50	86,20	79,50	7,00	0,30	6,70	10,77

Tabela 2.10: Wyniki zawierające modele uważane za nieprzeuczone uzyskane przy pomocy algorytmu 2.1.3.1 dla wyników przedstawionych na rysunku 2.5. Model 10 uznany jest jako najlepszy. Źródło: opracowanie własne

2.2 Programowanie genetyczne

Programowanie genetyczne jest częścią rodziny algorytmów ewolucyjnych, które ma na celu rozwiązywanie problemów, najczęściej optymalizacyjnych, wzorując się na procesie ewolucji naturalnej. Teoretycznym fundamentem algorytmów genetycznych jest twierdzenie o schematach [Rut09, s. 244-250]. Pomysł programowania genetycznego został przedstawiony w 1985 roku przez Michaela Lynn Cramera [Cra85], a spopularyzowany przez Johna Kozę dzięki monografii [Koz92], która obok teorii, zawierała implementację przedstawionej idei w języku LISP. W odróżnieniu od klasycznego algorytmu genetycznego przestrzenie genotypów i przestrzenie fenotypów nie są rozróżnialne. Dodatkowo, w programowaniu genetycznym rozwiązaniami problemu są programy reprezentowane przez genotypy, które należy uruchomić, aby ocenić ich działanie.

Jednym z najbardziej znanych rozwiązań nietrywialnego problemu przy pomocy programowania genetycznego jest skonstruowanie anteny o specyficznych parametrach na potrzeby misji kosmicznej NASA „Space Technology 5" [GHLL06]. W kontekście klasyfikacji obrazów wartą uwagi jest publikacja [SLL14]. Zaprezentowany jest tam algorytm ekstrakcji cech z wykorzystaniem programowania genetycznego. Miara skuteczności klasyfikacji uzyskana poprzez weryfikację z wykorzystaniem walidacji krzyżowej na uzyskanych deskryptorach numerycznych cech przy zastosowanym prostym klasyfikatorze jest lepsza od uczenia z wykorzystaniem deskryptorów numerycznych cech uzyskanych przy pomocy sieci neuronowych o architekturach DBN i CNN zakodowanych w ich warstwach ukrytych.

Z uwagi na wzorowanie się na procesie ewolucji naturalnej w algorytmach ewolucyjnych definiuje się następujące pojęcia:

- populacja P_e to zbiór osobników o określonym rozmiarze,
- osobnik O_e populacji P_e to „zakodowany w postaci chromosomu zbiór parametrów zadania, czyli rozwiązania" [Rut09, s. 231],
- chromosom CH_e to uporządkowane struktury genów (np. ciągi, drzewa, grafy),
- gen to G_e element, który „stanowi pojedynczy element genotypu, w szczególności chromosomu" [Rut09, s. 231],
- genotyp GE_e to „zespół chromosomów danego osobnika. Zatem osobnikami populacji mogą być genotypy albo pojedyncze chromosomy (jeśli genotyp składa się tylko z jednego chromosomu, a tak się często przyjmuje)" [Rut09, s. 231],
- fenotyp FE_e to „zestaw wartości odpowiadających danemu genotypowi, czyli zdekodowana struktura, a więc zbiór parametrów zadania (rozwiązanie, punkt przestrzeni poszukiwań)" [Rut09, s. 231],
- allel A_e to wartość danego genu,

2.2.1 Reprezentacja

Genotyp to zespół chromosomów danego osobnika populacji. Chromosomem nazywa się uporządkowane struktury genów. Na geny składają się wartości „elementarnych komponentów" [Fla11, s. 70]

zawarte w dwóch zbiorach [PLM08, s. 19-21]:

- F_S to zbiór funkcyjny (nieterminalny) zawierający operacje arytmetyczne, funkcje, „rozmaite instrukcje dostępne w danym języku programowania" [Fla11, s. 70] lub te charakterystyczne dla rozwiązywanego problemu,
- T_S to zbiór terminalny zawierający zmienne, stałe, operacje bezargumentowe.

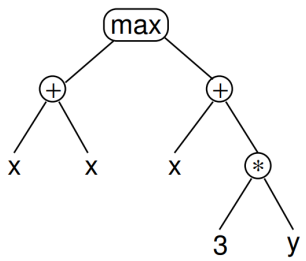
Każdy zbiór F_S powinien spełniać własność domknięcia (ang. closure) [PLM08, s. 21-22], którą rozumie się zapewnienie spójności typów danych w celu bezpiecznej ewaluacji programów, ponieważ operatory krzyżowania i mutacji prowadzą do losowych zmian w reprezentacji programu, a efektem niepożądanym są syntaktycznie nieprawidłowe programy. Tą własność można osiągnąć poprzez zastosowanie gramatyk formalnych [Whi95]. Zbiory F_S i T_S powinny spełniać własność wystarczalności (ang. sufficiency) [PLM08, s. 23]. Zbiory F_S i T_S są wystarczalne, jeżeli rozwiązywany problem da się opisać za pomocą sumy wartości ich zbiorów. Przykładem niewystarczającej sumy zbiorów F_S i T_S dla problemu wyrażeń arytmetycznych, w którym występuje mnożenie jest zbiór $\{-, x, y\}$.

Zbiór funkcyjny	
Komponent	*Wartości*
Arytmetyczny	$+$, $*$, $/$
Matematyczny	*sin*, *cos*, *exp*, *max*, *min*

Zbiór terminalny	
Komponent	*Wartości*
Zmiennych	x, y
Stałych	3, 0.45

Tabela 2.11: Tabele reprezentująca przykładowe zbiory funkcyjne i terminalne wraz z ich wartościami. Źródło: Opracowanie własne na podstawie [PLM08, s. 21]

Jeżeli chromosom reprezentowany jest za pomocą drzewa lub grafu, to wtedy jego wierzchołki będą etykietowane wartościami zbiorów F_S i T_S. Na rysunku 2.6 znajduje się przykładowy genotyp, w którego skład wchodzi jeden chromosom, będący drzewem syntaktycznym reprezentującym wyrażenie matematyczne $max(x + x, x + 3 * y)$.



Rysunek 2.6: Przykładowy genotyp reprezentujący wyrażenie arytmetyczne $max(x + x, x + 3 * y)$. Źródło: [PLM08, s. 10]

W celu wygenerowania populacji początkowej należy stworzyć jej osobniki. Klasycznie dla osobników, których genotyp składa się z jednego chromosomu o strukturze drzewa wyróżnia się trzy sposoby inicjalizacji z ograniczeniem maksymalnej wysokości struktury [PLM08, s. 11-14]:

- pełna (ang. full), gdzie jedynymi liścmi w drzewie są te na poziomie maksymalnej wysokości. Liście są etykietowane wartościami ze zbioru T_S. Wierzchołki wewnętrzne są etykietowane wartościami ze zbioru F_S,
- wzrostowa (ang. grow), gdzie liście mogą występować na każdej wysokości i są etykietowane wartościami ze zbioru T_S. Wierzchołki wewnętrzne są etykietowane wartościami ze zbioru F_S,
- pół na pół (ang. ramped half-and-half), gdzie połowa populacji początkowej jest inicjalizowana metodą wzrostową, a połowa metodą pełną.

2.2.2 Funkcja dopasowania

Każdy osobnik z populacji jest poddawany ocenie pod względem uzyskanego wyniku w kontekście rozwiązywanego problemu. W tym celu definiuje się funkcję dopasowania (przystosowania) F_e, jako:

- $F_e : P_e \rightarrow \mathbb{R}^+ \cup \{0\}$

Głównym celem algorytmów ewolucyjnych jest maksymalizacja lub minimalizacja[1] funkcji dopasowania. Maksymalizację stosuje się wtedy, kiedy zakłada się, że im wyższa wartość funkcji F_e dla danego osobnika z populacji, tym bardziej jest on przystosowany i uzyskane rozwiązanie jest lepsze (np. maksymalizacja miary F w problemie klasyfikacji). Minimalizację stosuje się, gdy dokonano założenia, że im niższa wartość funkcji F_e dla danego osobnika z populacji, tym bardziej jest on przystosowany (np. minimalizacja stopy błędu w problemie klasyfikacji).

2.2.3 Selekcja

Selekcja (reprodukcja) to proces, który ma na celu wybranie grupy osobników (populacji rodzicielskiej) będących dobrymi kandydatami do ich powielenia i utworzenia w ten sposób kolejnej populacji (populacji potomków) w celu zwiększania średniej wartości funkcji przystosowania osobników z populacji na populację. Klasycznie, proces realizuje się przy pomocy algorytmów takich jak:

- deterministyczny algorytm selekcji turniejowej [Rut09, s.270],
- algorytm selekcji rankingowej [Rut09, s.270-271],
- algorytm metody koła ruletki [Rut09, s.271],
- algorytm stochastycznego próbkowania uniwersalnego będącym uogólnieniem metody selekcji za pomocą metody koła ruletki.

Proces selekcji zapewnia, że algorytm ewolucyjny „ma tendencję podążania w kierunku lepszych rozwiązań. Ta zdolność poprawiania średniej wartości przystosowania populacji rodzicielskiej nazywa się naciskiem selektywnym. Mówimy, że algorytm charakteryzuje się dużym naciskiem selektywnym, gdy większa jest wartość oczekiwana liczby kopii lepszego osobnika niż wartość oczekiwana liczby

[1] Warto pamiętać, że problem minimalizacji w łatwy sposób można sprowadzić do problemu maksymalizacji.

kopii gorszego osobnika" [Rut09, s. 269]. Zmniejszając nacisk selektywny osiąga się eksplorację, czyli proces przeszukiwania przestrzeni rozwiązań, zorientowany na odnalezienie ekstremum globalnego – „wybierane są osobniki nie najlepiej przystosowane, ale mogące w przyszłości przybliżyć nas do optymalnego rozwiązania" [Rut09, s. 269]. Zwiększając nacisk selektywny osiąga się eksploatację, czyli proces przeszukiwania przestrzeni rozwiązań zorientowany na odnalezienie dowolnie dobrego ekstremum lokalnego lub globalnego – „osobniki przechodzące do następnej populacji mają wartości coraz bliższe oczekiwanego, choć niekoniecznie globalnego rozwiązania" [Rut09, s. 269].

2.2.4 Operatory genetyczne

Zważywszy na postać chromosomów, klasyczne implementacje operatorów krzyżowania i mutacji oparte na operacjach, na ciągach złożonych z zer i jedynek nie mają zastosowania w programowaniu genetycznym. Są one zastąpione specyficznymi operacjami na drzewach lub grafach. Procesy krzyżowania i mutacji zapewniają większe zróżnicowanie wśród osobników populacji, co „prowadzi do poszerzenia obszaru poszukiwań i zapobiega przedwczesnej zbieżności" [Rut09, s.269].

2.2.4.1 Operatory krzyżowania

Krzyżowanie to proces, w którym udział biorą najczęściej dwa losowo wybrane osobniki z populacji rodzicielskiej. Podczas krzyżowania tworzeni są ich potomkowie, najczęściej dwa, jeśli w krzyżowaniu brały udział dwa osobniki, na których z prawdopodobieństwem p_m wykonuje się operacje krzyżowania. Operatory można podzielić na homologiczne i niehomologiczne. Operatory homologiczne to takie, które zapewniają, że geny potomka będą rozmieszczone tak samo jak u jego rodziców. W literaturze obok operatora krzyżowania polegającego na przepinaniu poddrzew [Pol01], wyróżnia między innymi następujące rodzaje operatorów krzyżowania [PLM08, s. 44-46]:

- krzyżowanie jednopunktowe (ang. one-point crossover) będące operatorem homologicznym,
- krzyżowanie wielopunktowe (ang. multiple-point crossover) będące operatorem homologicznym,
- krzyżowanie jednorodne (ang. uniform crossover),
- krzyżowanie kontekstowe (ang. context-preversing crossover),
- krzyżowanie zorientowane na wysokość (ang. size-fair crossover).

2.2.4.2 Operatory mutacji

Mutacja to proces, który z prawdopodobieństwem p_m wprowadza losowe zmiany u osobników w populacji. W literaturze wyróżnia się między innymi następujące rodzaje operatorów krzyżowania [PLM08, s.42-44]:

- mutacja usunięcia całego poddrzewa i zastąpienia go losowym poddrzewem (ang. subtree mutation),
- mutacja zorientowana na wysokość (ang. size-fair subtree mutation),

- mutacja kurcząca (ang. shrink mutation),
- mutacja permutująca (ang. permutation mutation),
- mutacja wierzchołkowa (ang. node replacement mutation),
- mutacja stałych bazująca na losowości (ang. mutating constants at random).

2.2.5 Elitaryzm

Wybierając eksplorację poprzez zmniejszenie nacisku selektywnego istnieje duża szansa, że najlepiej przystosowane osobniki nie znajdą się w populacji potomków. Elitaryzm polega na ochronie z góry określonej liczby najlepiej przystosowanych osobników poprzez automatyczne włączenie takich osobników do populacji potomków.

2.2.6 Adaptacja

Adaptacja to zdolność algorytmu ewolucyjnego do kontrolowania swojego zachowania na podstawie statystyk jego działania i zdefiniowanych reguł rozmytego systemu wnioskującego. Przykładowo, tak jak w [Rut09, s. 289] możemy zdefiniować miarę α będącą ilorazem wyniku średniej wartości funkcji przystosowania w populacji przez wartość funkcji przystosowania dla najlepszego osobnika w populacji i zdefiniować opartą na niej regułę „jeżeli α jest duże to zwiększ populację" [Rut09, s. 289].

2.2.7 Przebieg algorytmu

W celu optymalizacji i stworzenia populacji początkowej programów potrzeba: zdefiniować cel procesu programowania genetycznego, zbiór funkcji, zbiór terminalny, funkcję dopasowania, algorytm selekcji, parametry takie jak liczebność populacji początkowej i strategię jej generowania, stosowanie elitaryzmu i liczebność elitarnych osobników, prawdopodobieństwa krzyżowania i mutacji, ewentualne limity co do złożoności chromosomu oraz kryterium stopu.

Przebieg klasycznego algorytmu programowania genetycznego z uwzględnieniem elitaryzmu i adaptacji jest następujący:

1. Stwórz początkową populację osobników.
2. Oblicz wartość funkcji przystosowania dla każdego osobnika populacji. W tym celu uruchom powstałe programy.
3. Sprawdź kryterium stopu.
4. Dokonaj selekcji osobników:
    - jeśli wybrano stosowanie strategii elitarnej, to włącz daną liczbę najlepiej przystosowanych osobników do następnej populacji, a potem dokonaj selekcji przy pomocy wybranego algorytmu,

- w przeciwnym razie dokonaj selekcji przy pomocy wybranego algorytmu.

5. Zastosuj operatory genetyczne z zadanym prawdopodobieństwem.
6. W wyniku zastosowania selekcji i operatorów genetycznych powstała nowa populacja.
7. Wróć do sprawdzenia kryterium stopu.
   - jeśli stop, to wskaż najlepszego osobnika,
   - w przeciwnym razie sprawdź miary adaptacyjnego systemu wnioskującego, zastosuj reguły i przejdź do punktu 4.

2.3 Gramatyka grafowa

Gramatyka formalna to sposób opisu języka formalnego. Językiem generowanym przez gramatykę będzie dowolny podzbiór zbioru wszystkich słów nad ustalonym alfabetem, które zostały wyprowadzone z aksjomatu (symbolu początkowego) przy zastosowaniu zdefiniowanych produkcji (reguł przepisujących struktury). W zależności od rozważanej gramatyki, Mariusz Flasiński w swojej monografii [Fla11] wyróżnia podział gramatyk na: ciągowe, drzewowe i grafowe, alfabetem jest skończony zbiór symboli, wierzchołków lub krawędzi. Zatem słowo to skończony ciąg symboli, drzewo bądź graf.

W kolejnych podrozdziałach zostały przedstawione definicje grafu prostego, grafu atrybutowanego, produkcji, diagramu sterującego oraz definicja atrybutowanej programowalnej gramatyki grafowej. System ten jest autorstwa Horsta Bunkego [Bun83] [Bun82].

2.3.1 Graf prosty

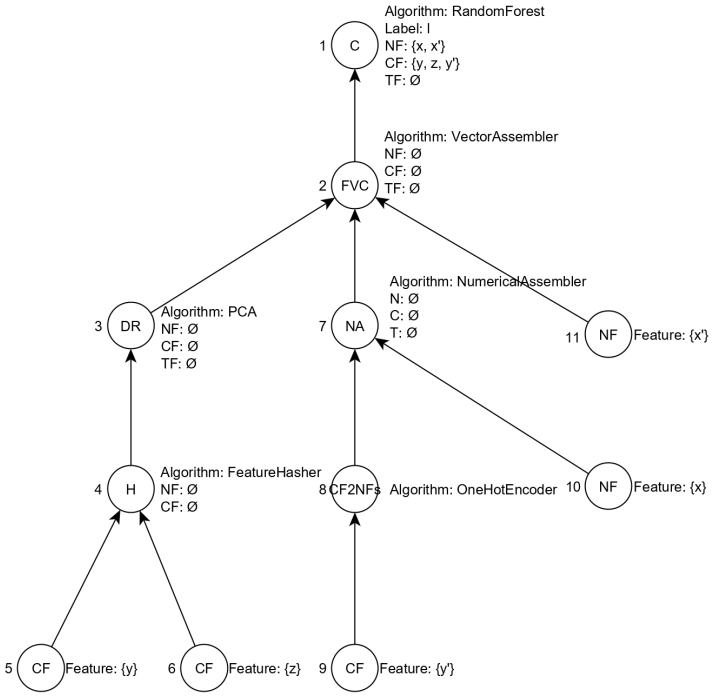
Graf prosty $(u - graf)$ jako trójkę $g = (N, E, \lambda)$, gdzie:

- N to skończony zbiór wierzchołków,
- $E = (E_w)_{w \in W}$ to krotką relacji $E_w \in N \times N$ dla każdej krawędzi $w \in W$,
- λ to funkcja etykietowania wierzchołków taka, że $\lambda : N \to V$,

2.3.2 Graf atrybutowany

Graf atrybutowany $(a - graf)$ jest piątką $g = (N, E, \lambda, \alpha, \beta)$, gdzie:

- N zdefiniowane tak samo jak w grafie prostym,
- E zdefiniowane tak samo jak w grafie prostym,
- λ zdefiniowane tak samo jak w grafie prostym,
- α to funkcja przypisująca zbiór atrybutów wierzchołkowych każdemu wierzchołkowi $n \in N$ taka, że $\alpha : N \to 2^A$,
- β to funkcja przypisująca zbiór atrybutów krawędziowych każdej krawędzi $w \in W$ taka, że $\beta = (\beta_W)_{w \in W}$, gdzie $\beta_W : E_W \to 2^B$.



Rysunek 2.7: Przykładowe drzewo będące grafem atrybutowanym reprezentujące model klasyfikujący. Źródło: opracowanie własne

2.3.3 Produkcja

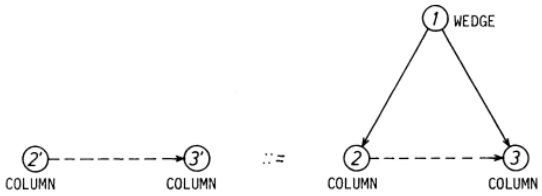
Atrybutowany graf g' jest podgrafem grafu atrybutowanego g, jeśli wszystkie wierzchołki i krawędzie grafu g' należą do grafu g. Dodatkowo odpowiadające sobie wierzchołki i krawędzie muszą mieć takie same etykiety i atrybuty. Dla danych grafów g, g' takich, że g' jest podgrafem g, przez $g - g'$ oznaczamy graf, który powstaje przez usunięcie g' z g. Krawędzie pomiędzy grafem g' a grafem $g - g'$ nazywamy osadzeniem g' w $g - g'$ i oznaczamy $EMB(g', g)$

Produkcja to piątka $p = (g_l, g_r, T, \pi, F)$, gdzie

1. g_l to graf atrybutowany (tak zwana lewa strona produkcji),
2. g_r to graf atrybutowany (tak zwana prawa strona produkcji),
3. T to transformacja osadzenia definującą sposób zastąpienia krawędzi $EMB(G, H)$, gdzie G jest izomorficznym obrazem g_l, H jest grafem z którego G został usunięty. $T = \{(L_w, R_w) | w \in W\}$, gdzie $(L_w, R_w) \in N_l \times N_r$,$N_l$ i N_r są wierzchołkami grafów izomorficznych z g_l i g_r,
4. π to predykat stosowalności produkcji, taki, że $\pi : \Gamma \rightarrow \{true, false\}$, gdzie Γ jest rodziną wszystkich grafów atrybutowanych,
5. f jest skończonym zbiorem funkcji częściowych, przenoszących atrybuty wierzchołkowe f_a i krawędziowe f_b, takich, że $f_a : N_r \rightarrow D_a$ i $f_b : E_r \rightarrow D_b$, gdzie $a \in A$ i $b \in B$.

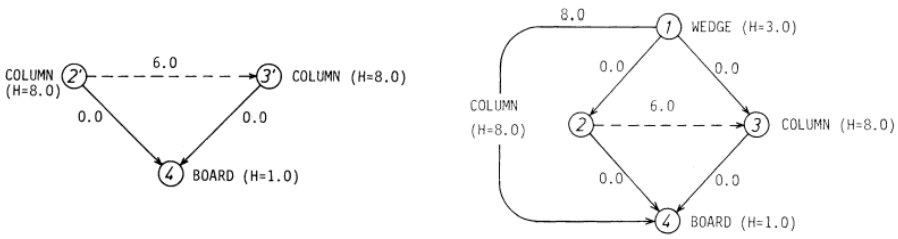
Bezpośrednie wyprowadzenie grafu g' z g przy użyciu p ($g \xrightarrow{p} g'$) jest możliwe gdy g_l jest podgrafem g i predykat stosowalności pozwala na zastosowanie produkcji (wynikiem testu predykatu jest wartość *true*). Jeśli oba warunki są spełnione dalsze kroki procedury są następujące:

- zastąp graf lewej strony produkcji grafem prawej strony produkcji stosując generyczną transformację osadzenia,
- zastosuj funkcje f_a i f_b w celu przeniesienia atrybutów wierzchołkowych i krawędziowych do g_r, który zastępuje g_l.



π: HEIGHT (2′) = HEIGHT (3′).
$T = \{R_{\text{BELOW}}\}$ with $R_{\text{BELOW}} = \{(2', 2), (3', 3), (2', 1)\}$.
$F = \{f_H, f_{\text{DIS}}\}$ with H = HEIGHT, DIS = DISTANCE:
$f_H(2) = H(2'); f_H(3) = H(3'); f_H(1) = 3.0$.
$f_{\text{DIS}}((2, 3)_{\text{RIGHT-OF}}) = \text{DIS}\,((2', 3')_{\text{RIGHT-OF}})$,
$f_{\text{DIS}}((1, 2)_{\text{BELOW}}) = f((1, 3)_{\text{BELOW}}) = 0.0$,
$f_{\text{DIS}}((2, -)_{\text{BELOW}}) = f_{\text{DIS}}((3, -)_{\text{BELOW}}) = 0.0$,
$f_{\text{DIS}}((1, -)_{\text{BELOW}}) = H(2)$.

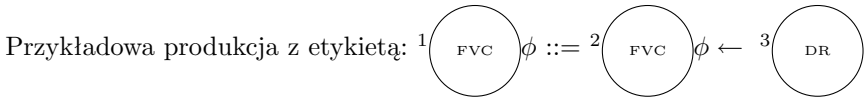
Rysunek 2.8: Przykładowa produkcja zawierająca lewą stronę produkcji, prawą stronę produkcji, predykat stosowalności, funkcje przenoszące atrybuty oraz transformację osadzenia.
Źródło: [Bun82, s. 575-576]

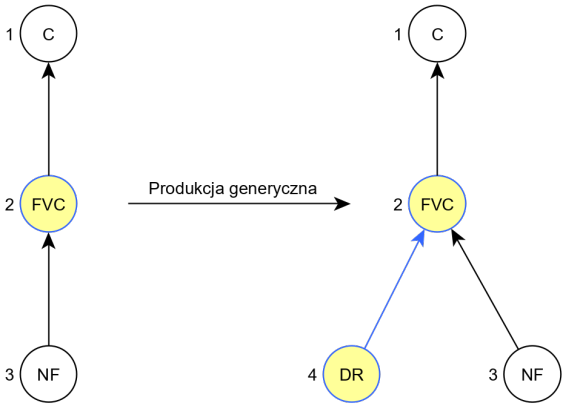


Rysunek 2.9: Dwa przykładowe atrybutowane grafy reprezentujące budynek.
Źródło: [Bun82, s. 575-576]

Dla zobrazowania zastosowania produkcji niech posłuży powyższy rysunek 2.9. Znajdują się na nim dwa atrybutowane grafy. Ten po lewej stronie rysunku, to graf do którego zostanie zastosowana produkcja z rysunku 2.8. Graf po prawej stronie to efekt zastosowania produkcji, transferu atrybutów i transformacji osadzenia.

Dodatkowo dla produkcji wprowadza się pomocniczą notację mającą na celu pomóc w śledzeniu osadzeń. Etykietą ϕ oznacza się jeden wierzchołek w g_l oraz jeden wierzchołek w g_r. Takie oznaczenie należy rozumieć, że wszystkie krawędzie w wychodzące z wierzchołka $n \in N$ i dochodzące do wierzchołka $n \in N$ oznaczonego etykietą ϕ w grafie g_l po jego zastąpieniu grafem g_r będą dochodzić do wierzchołka $n \in N$ i wychodzić z wierzchołka $n \in N$ oznaczonego etykietą ϕ. Dla objaśnienia niech posłuży poniższy produkcja i rysunek 2.10. Predykat stosowalności oraz oraz funkcje transferu atrybutów zostały celowo pominięte dla zobrazowania celu zastosowania notacji.





Rysunek 2.10: Zastosowanie produkcji. Źródło: opracowanie własne

Na powyższym rysunku 2.10 widać, że wszystkie krawędzie, które wychodziły $(2, 1)$ i dochodziły $(3, 2)$ do wierzchołka etykietowanego FVC nadal z niego wychodzą $(2, 1)$ i dochodzą $(3, 2)$ oraz dodatkowo dochodzi krawędź $(4, 2)$ w wyniku osadzenia prawej strony produkcji w grafie, którego podgraf był lewą stroną produkcji.

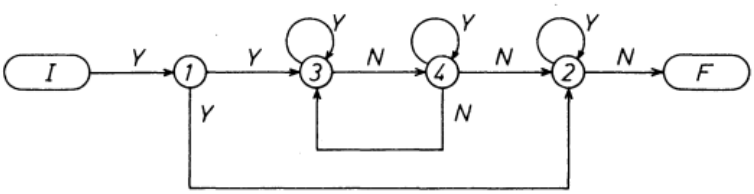
2.3.4 Diagram sterujący

Diagram sterujący CD nad P definiowany jest w celu określenia kolejności w jakiej powinny zostać zastosowane produkcje gramatyki. Diagram definiuje się jako graf prosty, gdzie dany jest zbiór etykiet wierzchołkowych taki, że $V = P \cup \{I, F\}$ oraz zbiór etykiet krawędziowych taki, że $W = \{Y, N\}$ oraz dodatkowo:

- istnieje tylko jeden wierzchołek początkowy n_I etykietowany etykietą I,
- istnieje tylko jeden wierzchołek końcowy n_F etykietowany etykietą F,
- nie istnieje krawędź dochodząca do wierzchołka n_I,

- nie istnieje krawędź wychodząca z wierzchołka n_F.

Zastosowanie diagramu sterującego sprowadza się do jego przejścia zaczynając od wierzchołka początkowego n_I kończąc na wierzchołku końcowym n_F. Produkcją początkową, którą należy zastosować jest ta z etykiety wierzchołka, do którego dochodzi krawędź etykietowana wartością Y i ta krawędź bezpośrednio wychodzi z n_I. Jeżeli zastosowanie produkcji powiodło się (predykat stosowalności został spełniony) należy przejść do wierzchołka etykietowanego tą produkcją wykorzystując krawędź Y. Jeżeli produkcji nie udało się zastosować należy przejść do wierzchołka etykietowanego produkcją wykorzystując krawędź N chyba, że istnieje możliwość zastosowania innej produkcji z powodzeniem. Wtedy należy zastosować właśnie taką produkcję i przejść do wierzchołka nią etykietowaną wykorzystując krawędź Y. Jeżeli z jednego wierzchołka wychodzi więcej niż jedna krawędź o tej samej etykiecie, to do następnego wierzchołka należy podążać losowo wybraną krawędzią.



Rysunek 2.11: Przykładowy diagram sterujący zaproponowany przez Horsta Bunkego.
Źródło: [Bun82, s. 576]

W przykładzie z powyższego rysunku 2.11 należy zacząć od zastosowania produkcji 1. Zakładając, że zastosowanie produkcji powiodło się, należy przejść z wierzchołka n_I do wierzchołka etykietowanego produkcją 1 z wykorzystaniem krawędzi Y. Z tego wierzchołka można osiągnąć wierzchołki etykietowane produkcją 2 i 3. Zakładając, że dla produkcji 2 i 3 jest spełniony predykat stosowalności i zostało wybrane losowo zastosowanie produkcji 3, to wierzchołek etykietowany produkcją 3 został osiągnięty poprzez krawędź Y. Zakładając, że produkcja 3 nie może zostać powtórnie zastosowana należy przejść do wierzchołka etykietowanego produkcją 4 wykorzystując krawędź N. Z tego wierzchołka można osiągnąć wierzchołki etykietowane produkcjami 2, 3, jeśli produkcja 4 nie będzie mogła zostać wykonana i 4. Zakładając, że produkcja 4 nie może zostać użyta i losowo zostało wybrane przejście do wierzchołka 2 krawędzią N oraz, że produkcja 2 nie może zostać zastosowana, to osiągnięto wierzchołek końcowy n_F przy użyciu krawędzi N. Wygenerowane w ten sposób słowo jest poprawne i należy do języka.

Na cele pracy dokonuje się założenia, że poprzez wierzchołek początkowy n_I można przekazać atrybuty początkowe, które zasilą graf startowy. Ma to na celu ułatwić zdefiniowanie produkcji dla rozważanego problemu, ponieważ dokonanie wyboru, które cechy z zbioru danych wejściowych powinny zostać wybrane do zbudowania modelu odbywa się poza systemem. Dodatkowo od tego miejsca zakłada się, że P w G i CD, to zbiór produkcji generycznych.

2.3.5 Atrybutowana programowalna gramatyka grafowa

W publikacjach dotyczących gramatyk grafowych autorstwa Horsta Bunkego [Bun83] [Bun82] definiuje się atrybutowaną, programowalną gramatykę grafową jako szczególny rodzaj gramatyki grafowej. Jest to siódemka $G = (V, W, A, B, P, S, C)$, gdzie:

- V i W to alfabety etykiet wierzchołków i krawędzi,
- A i B to skończone zbiory atrybutów wierzchołkowych i krawędziowych,
- P to skończony zbiór produkcji,
- S to skończony zbiór grafów początkowych,
- CD to diagram sterujący gramatyki.

Językiem generowanym przez gramatykę G będą wszystkie atrybutowane grafy, które mogą zostać wyprowadzone z grafu początkowego przy zastosowaniu produkcji P w kolejności zdefiniowanej w diagramie sterującym CD. Przez wyprowadzenie rozumie się stosowanie produkcji przechodząc diagram sterujący, zaczynając od wierzchołka początkowego kończąc na osiągnięciu wierzchołka końcowego.

Rozdział 3

Proponowane rozwiązanie

W celu generowania modeli klasyfikujących i ich optymalizacji autor proponuje rozwiązanie oparte o programowanie genetyczne i gramatykę grafową. Opracowanie języka i własnej gramatyki, która będzie go generować ma na celu dostarczenie narzędzia nie tylko do opisu modeli klasyfikujących, ale i poprawnego generowania osobników składających się na całą początkową populację modeli. Co więcej, dzięki proponowanej gramatyce zostaje rozwiązany problem zdefiniowania operatorów mutacji względem wierzchołków reprezentujących symbole nieterminalne z punktu widzenia programowania genetycznego. Powstałe w wyniku mutacji osobniki będą poprawne i będą należeć do zdefiniowanego języka. W kolejnych podrozdziałach znajdują się implementacje gramatyki grafowej jak i programowania genetycznego zgodne z definicjami przedstawionymi w rozdziale 2. Podrozdział „Algorytm jako całość" jest klamrą kompozycyjną w kontekście zaproponowanych w tym rozdziale rozwiązań. Jest w nim przedstawiony algorytm będący najważniejszą funkcjonalnością implementowanej biblioteki, której opis implementacyjny i architektoniczny jest zawarty w Dodatku A, a wyniki jej działania w rozdziale 4.

3.1 Implementacja gramatyki grafowej

W kolejnych podrozdziałach została opisana implementacja stworzonej gramatyki grafowej. Dodatkowo zostały zdefiniowane funkcje i pomocnicze operacje na zbiorach, które pozwolą skrócić zapis funkcji transferu atrybutów dla każdej ze zdefiniowanej produkcji.

3.1.1 Alfabety i zbiory atrybutów

Dla zdefiniowanej gramatyki w postaci tabelarycznej definiuje się następujące zbiory: alfabet etykiet wierzchołków, alfabet etykiet krawędzi, alfabet atrybutów wierzchołkowych i krawędziowych.

Zbiór	Wartości
V	S, C, ENS_C, FVC, DE, DR, H, NA, $NF2NF$, $NF2NF_S$, $CF2NF$, $CF2NF_S$, $TF2NF$, $TF2NF_S$, $TF2TF$, TA, TA_S, NF, CF, TF
W	brak wartości
A	$Algorithm$, $Classifiers$, $Label$, NF, CF, TF, $Feature$
B	brak wartości

Tabela 3.1: Zbiory alfabetu etykiet wierzchołków, alfabetu etykiet krawędzi, alfabetu atrybutów wierzchołkowych i krawędziowych. Źródło: opracowanie własne

W postaci tabelarycznej definiuje się zbiory atrybutów wierzchołkowych.

Zbiór	Wartości
ENS_{alg}	Majority Voting, Weighted Majority Voting, Naive Bayes Rule, Kulbecker Leibler Method, PCA Integration, Single Layered Neural Network Integration
C_{alg}	Naive Bayes, Logistic Regression, Decision Tree, Random Forest, Stochastic Gradient Boosted Decision Trees, Linear Support Vector Machine, Multilayer Perceptron
FVC_{alg}	Vector Assembler, Chi-Squared Based Feature Selector
DE_{alg}	Polynomial Expansion
DR_{alg}	PCA, Chi-Squared Based Feature Selector
H_{alg}	Murmur Hash
NA_{alg}	Numerical Assembler
NF2NF_{alg}	Standard Scaler, Normalizer, Min Max Scaler, Max Abs Scaler, Bucketizer, Binarizer, Imputer
CF2NF_{alg}	One Hot Encoder, String Indexer, Tokenizer, Count Vectorizer, Inverse Document Frequency
TF2NF_{alg}	Hashing Term Frequency, Hashing Term Frequency - IDF, Count Vectorizer - IDF, Word2Vec, Count Vectorizer
TF2TF_{alg}	nGram, Normalizer, Stemmer, Lemmatizer
TA_{alg}	Text Assembler

Tabela 3.2: Zbiory atrybutów wierzchołkowych i ich wartości. Źródło: opracowanie własne

3.1.2 Produkcje

3.1.2.1 Pomocnicze funkcje

Wszystkie zbiory wymienione w tabeli 3.2 zawierają się w zbiorze All_{alg}. Dla każdego zbioru X zawartego w zbiorze algorytmów All_{alg} definiuje się funkcję Λ_X przyporządkowywującą losowy element ze zbioru X dla każdej wartości etykiety, gdzie $\Lambda_X : V \rightarrow X$. Na przykład $\Lambda_{C_{alg}}(v)$, gdzie $v = C$, to wynikiem funkcji jest losowo wybrany element ze zbioru C_{alg} (np. Naive Bayes), który jest wartością zbioru C_{alg} i atrybutem wierzchołkowym reprezentującym nazwę algorytmu.

Dodatkowo definiuje się kolejną funkcję, gdzie:

- Ψ to funkcja przypisująca algorytm dla każdej etykiety wierzchołka $v \in V$ taka, że $\Psi_{alg} : V \to All_{alg}$ i:

$$\Psi_{alg}(v) = \begin{cases} \Lambda_{ENS_{C_{alg}}}(v), & \text{jeżeli } v \text{ to } ENC_C \\ \Lambda_{C_{alg}}(v), & \text{jeżeli } v \text{ to } C \\ \Lambda_{FVC_{alg}}(v), & \text{jeżeli } v \text{ to } FVC \\ \Lambda_{DE_{alg}}(v), & \text{jeżeli } v \text{ to } DE \\ \Lambda_{DR_{alg}}(v), & \text{jeżeli } v \text{ to } DR \\ \Lambda_{H_{alg}}(v), & \text{jeżeli } v \text{ to } H \\ \Lambda_{NA_{alg}}(v), & \text{jeżeli } v \text{ to } NA \\ \Lambda_{NF2NF_{alg}}(v), & \text{jeżeli } v \text{ to } NF2NF \text{ lub } NF2NF_S \\ \Lambda_{CF2NF_{alg}}(v), & \text{jeżeli } v \text{ to } CF2NF \text{ lub } CF2NF_S \\ \Lambda_{TF2NF_{alg}}(v), & \text{jeżeli } v \text{ to } TF2NF \text{ lub } TF2NF_S \\ \Lambda_{TF2TF_{alg}}(v), & \text{jeżeli } v \text{ to } TF2NF \text{ lub } TF2TF_S \\ \Lambda_{TA_{alg}}(v), & \text{jeżeli } v \text{ to } TA \text{ lub } TA_S \\ \varnothing & \text{w przeciwnym wypadku} \end{cases}$$

3.1.2.2 Pomocnicze operacje na wartościach atrybutów wierzchołkowych

Dla atrybutów wierzchołkowych $F \in \{NF, CF, TF\}$ ich wartościami są zbiory cech. Zostały dla nich zdefiniowane następujące operacje:

$$subset_F(n) = \begin{cases} F' : F' \subseteq F(n) \wedge |F'| \geqslant 0, & \text{jeżeli } |F(n)| > 0 \\ \varnothing & \text{w przeciwnym wypadku} \end{cases}$$

$$subset'_F(n) = \begin{cases} F' : F' \subset F(n) \wedge |F'| = 1, & \text{jeżeli } |F(n)| > 0 \\ \varnothing & \text{w przeciwnym wypadku} \end{cases}$$

Przykładowo, jeżeli wartością atrybutu wierzchołkowego NF wierzchołka 1 jest zbiór dwuelementowy $\{x, y\}$, czyli $NF(1) = \{x, y\}$, to w wyniku operacji $subset_{NF}(1)$ możliwymi wynikami są podzbiory $\{x\}$, $\{y\}$, $\{x, y\}$ lub $\varnothing$. W wyniku operacji $subset'_{NF}(1)$ możliwymi wynikami są podzbiory $\{x\}$, lub $\{y\}$.

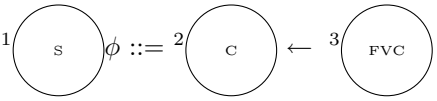
3.1.2.3 Zdefiniowane produkcje

Zastosowaną notację w funkcji transferu atrybutów kolejnych produkcji należy rozumieć w następujący sposób:

- $f_a(n)$ określa wartość atrybutu a w wierzchołku n prawej strony produkcji,
- $a(n)$ określa wartość atrybutu a w wierzchołku n,

- $v(n)$ określa etykietę wierzchołka n,
- $\Psi_{alg}(v(n))$ określa wartość funkcji Ψ_{alg} dla etykiety wierzchołka n,
- $subset_F(n)$, $subset'_F(n)$ określa podzbiór zbioru będącego wartością dla atrybutu wierzchołkowego $F \in \{NF, CF, TF\}$ w wierzchołku n zgodnie ze zdefiniowaną wyżej operacją.

Produkcja 1.1

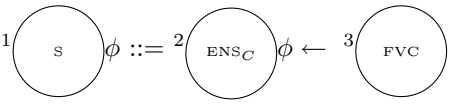


π : *true*

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \Psi_{alg}(v(2))$
$f_{Label}(2) = \text{Label}(1)$
$f_{NF}(3) = \text{NF}(1)$
$f_{CF}(3) = \text{CF}(1)$
$f_{TF}(3) = \text{TF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1)$
$f_{TF}(2) = \text{TF}(1)$

Produkcja 2.1

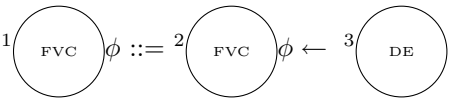


π : *true*

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \Psi_{alg}(v(2))$
$f_{Classifiers}(2) = \{$
$\quad \Psi_{alg}(v(2)),$
$\quad \Psi_{alg}(v(2)),$
$\quad \Psi_{alg}(v(2))$
$\}$
$f_{Label}(2) = \text{Label}(1)$
$f_{NF}(3) = \text{NF}(1)$
$f_{CF}(3) = \text{CF}(1)$
$f_{TF}(3) = \text{TF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1)$
$f_{TF}(2) = \text{TF}(1)$

Produkcja 4.1

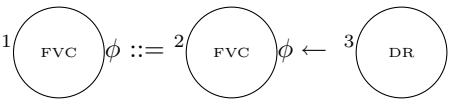


π :
$|NF(1)| > 0 \vee |CF(1)| > 0 \vee |TF(1)| > 0$

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{NF}(3) = subset_{NF}(1)$
$f_{CF}(3) = subset_{CF}(1)$
$f_{TF}(3) = subset_{TF}(1)$
$f_{NF}(2) = \text{NF}(1) \setminus \text{NF}(3)$
$f_{CF}(2) = \text{CF}(1) \setminus \text{CF}(3)$
$f_{TF}(2) = \text{TF}(1) \setminus \text{TF}(3)$

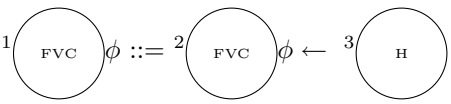
Produkcja 4.2



π : predykat taki jak w 4.1

F : funkcje takie jak w 4.1

Produkcja 4.3



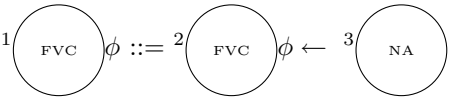
π : $|NF(1)| > 0 \vee |CF(1)| > 0$

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{NF}(3) = subset_{NF}(1)$

$f_{CF}(3) = subset_{CF}(1)$
$f_{NF}(2) = \text{NF}(1) \setminus \text{NF}(3)$
$f_{CF}(2) = \text{CF}(1) \setminus \text{CF}(3)$
$f_{TF}(2) = \text{TF}(1)$

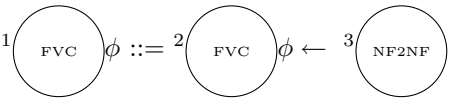
Produkcja 4.4



π : predykat taki jak w 4.1

F : funkcje takie jak w 4.1

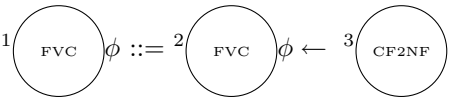
Produkcja 4.5



$\pi : |NF(1)| > 0$

F :
$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{NF}(3) = subset'_{NF}(1)$
$f_{NF}(2) = \text{NF}(1) \setminus \text{NF}(3)$
$f_{CF}(2) = \text{CF}(1)$
$f_{TF}(2) = \text{TF}(1)$

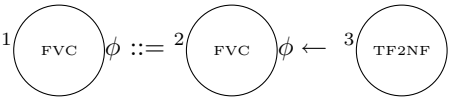
Produkcja 4.6



$\pi : |CF(1)| > 0$

F :
$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{CF}(3) = subset'_{CF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1) \setminus \text{CF}(3)$
$f_{TF}(2) = \text{TF}(1)$

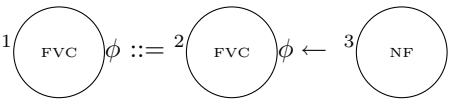
Produkcja 4.7



$\pi : |TF(1)| > 0$

F :
$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{TF}(3) = subset_{TF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1)$
$f_{TF}(2) = \text{TF}(1) \setminus \text{TF}(3)$

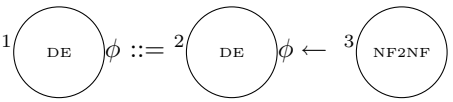
Produkcja 4.8



$\pi : |NF(1)| > 0$

F :
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{Feature}(3) = subset'_{NF}(1)$
$f_{NF}(2) = \text{NF}(1) \setminus \text{Feature}(3)$
$f_{CF}(2) = \text{CF}(1)$
$f_{TF}(2) = \text{TF}(1)$

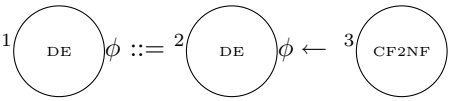
Produkcja 5.1



π : predykat taki jak w 4.5

F : funkcje takie jak w 4.5

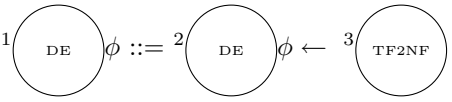
Produkcja 5.2



π : predykat taki jak w 4.6

F : funkcje takie jak w 4.6

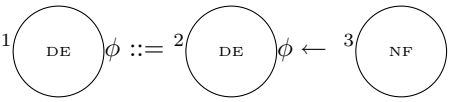
Produkcja 5.3



π : predykat taki jak w 4.7

F : funkcje takie jak w 4.7

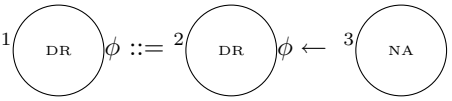
Produkcja 5.4



π : predykat taki jak w 4.8

F : funkcje takie jak w 4.8

Produkcja 6.1



π : predykat taki jak w 4.1

F : funkcje takie jak w 4.1

Produkcja 6.2

π : predykat taki jak w 4.1

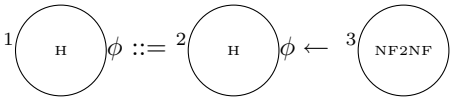
F : funkcje takie jak w 4.1

Produkcja 6.3

π : predykat taki jak w 4.3

F : funkcje takie jak w 4.3

Produkcja 7.1

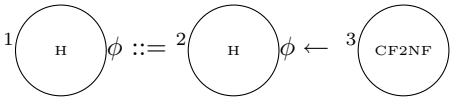


π : $|NF(1)| > 0$

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{NF}(3) = subset'_{NF}(1)$
$f_{NF}(2) = \text{NF}(1) \setminus \text{NF}(3)$
$f_{CF}(2) = \text{CF}(1)$

Produkcja 7.2

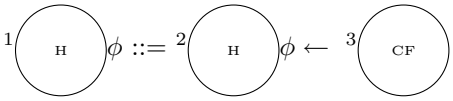


π : $|CF(1)| > 0$

F :

$f_{Algorithm}(3) = \Psi_{alg}(v(3))$
$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{CF}(3) = subset'_{CF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1) \setminus \text{CF}(3)$

Produkcja 7.3

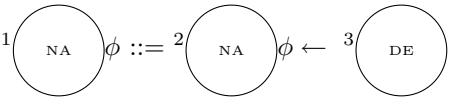


π : $|CF(1)| > 0$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{Feature}(3) = subset'_{CF}(1)$
$f_{NF}(2) = \text{NF}(1)$
$f_{CF}(2) = \text{CF}(1) \setminus \text{Feature}(3)$

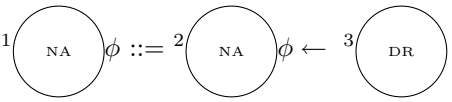
Produkcja 8.1



π : predykat taki jak w 4.1

F : funkcje takie jak w 4.1

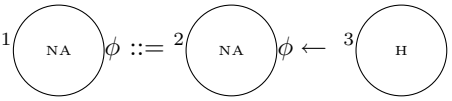
Produkcja 8.2



π : predykat taki jak w 4.1

F : funkcje takie jak w 4.1

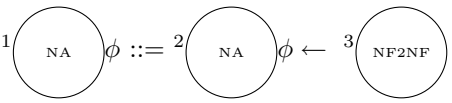
Produkcja 8.3



π : predykat taki jak w 4.3

F : funkcje takie jak w 4.3

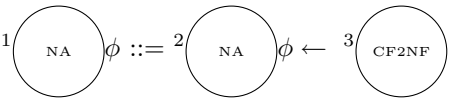
Produkcja 8.4



π : predykat taki jak w 4.5

F : funkcje takie jak w 4.5

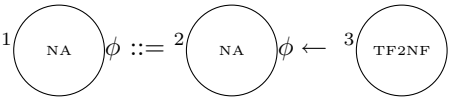
Produkcja 8.5



π : predykat taki jak w 4.6

F : funkcje takie jak w 4.6

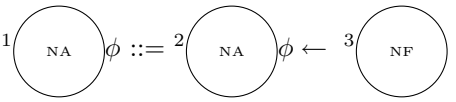
Produkcja 8.6



π : predykat taki jak w 4.7

F : funkcje takie jak w 4.7

Produkcja 8.7



π : predykat taki jak w 4.8

F : funkcje takie jak w 4.8

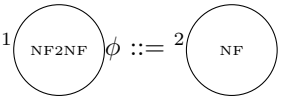
Produkcja 9.1

π : $|NF(1)| = 1$

F :

$f_{Algorithm}(2)$ = Algorithm(1)
$f_{Feature}(3)$ = NF(1)

Produkcja 9.2

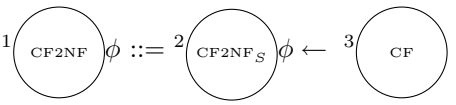


π : $|NF(1)| = 1$

F :

$f_{Algorithm}(2)$ = Algorithm(1)
$f_{Feature}(2)$ = NF(1)

Produkcja 10.1



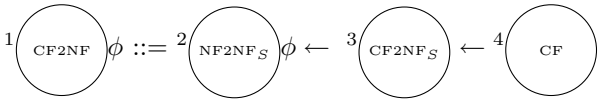
π : $|CF(1)| = 1$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$

$f_{Feature}(3) = \text{CF}(1)$

Produkcja 10.2



$\pi : |CF(1)| = 1$

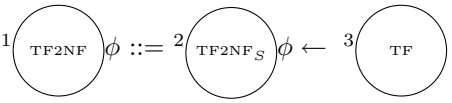
F :

$f_{Algorithm}(3) = \text{Algorithm}(1)$

$f_{Algorithm}(2) = \Psi(v(2))$

$f_{Feature}(4) = \text{CF}(1)$

Produkcja 11.1



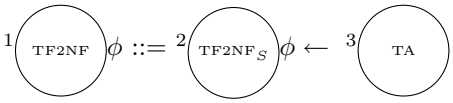
$\pi : |TF(1)| = 1$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$

$f_{Feature}(3) = TF(1)$

Produkcja 11.2



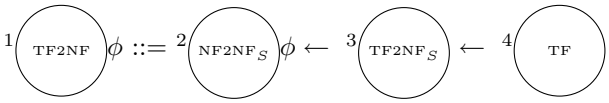
$\pi : |TF(1)| > 0$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$

$f_{Feature}(3) = subset_{TF}(1)$

Produkcja 11.3



$\pi : |TF(1)| = 1$

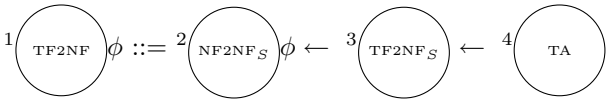
F :

$f_{Algorithm}(3) = \text{Algorithm}(1)$

$f_{Algorithm}(2) = \Psi_{alg}(v(2))$

$f_{Feature}(4) = TF(1)$

Produkcja 11.4

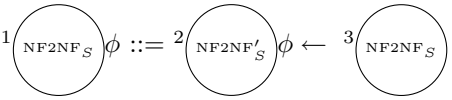


$\pi : |TF(1)| > 0$

F :

$f_{Algorithm}(3) = \text{Algorithm}(1)$
$f_{Algorithm}(2) = \Psi_{alg}(v(2))$
$f_{NF}(4) = subset_{TF}(1)$

Produkcja 12.1



$\pi : true$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{Algorithm}(3) = \Psi_{alg}(v(2))$

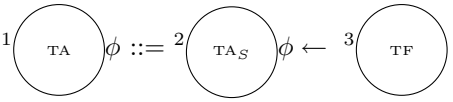
Produkcja 13.1

$\pi : true$

F :

$f_{Algorithm}(3) = \text{Algorithm}(1)$
$f_{Feature}(3) = \text{Feature}(1)$
$f_{Algorithm}(2) = \Psi_{alg}(v(2))$

Produkcja 14.1

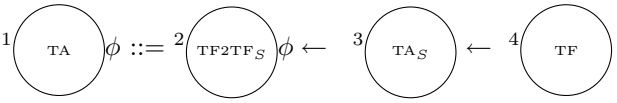


$\pi : |TF(1)| > 0$

F :

$f_{Algorithm}(2) = \text{Algorithm}(1)$
$f_{Feature}(3) = subset'_{TF}(1)$
$f_{TF}(2) = \text{TF}(1) \setminus \text{Feature}(3)$

Produkcja 14.2

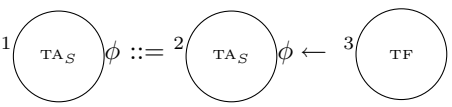


$\pi : |TF(1)| > 0$

F :

$f_{Algorithm}(3) = \text{Algorithm}(1)$
$f_{Algorithm}(2) = \Psi_{alg}(v(2))$
$f_{Feature}(4) = subset'_{TF}(1)$
$f_{TF}(3) = \text{TF}(1) \setminus \text{Feature}(4)$

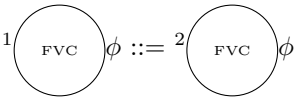
Produkcja 15.1



π : predykat taki jak w 14.1

F : funkcje takie jak w 14.1

Produkcja 16.1



π :
$|NF(1)| > 0 \vee |CF(1)| > 0 \vee |TF(1)| > 0$

F:
$f_{Algorithm}(2)$ = Algorithm(1)
$f_{NF}(2)$ = NF(1)
$f_{CF}(2)$ = CF(1)
$f_{TF}(2)$ = TF(1)

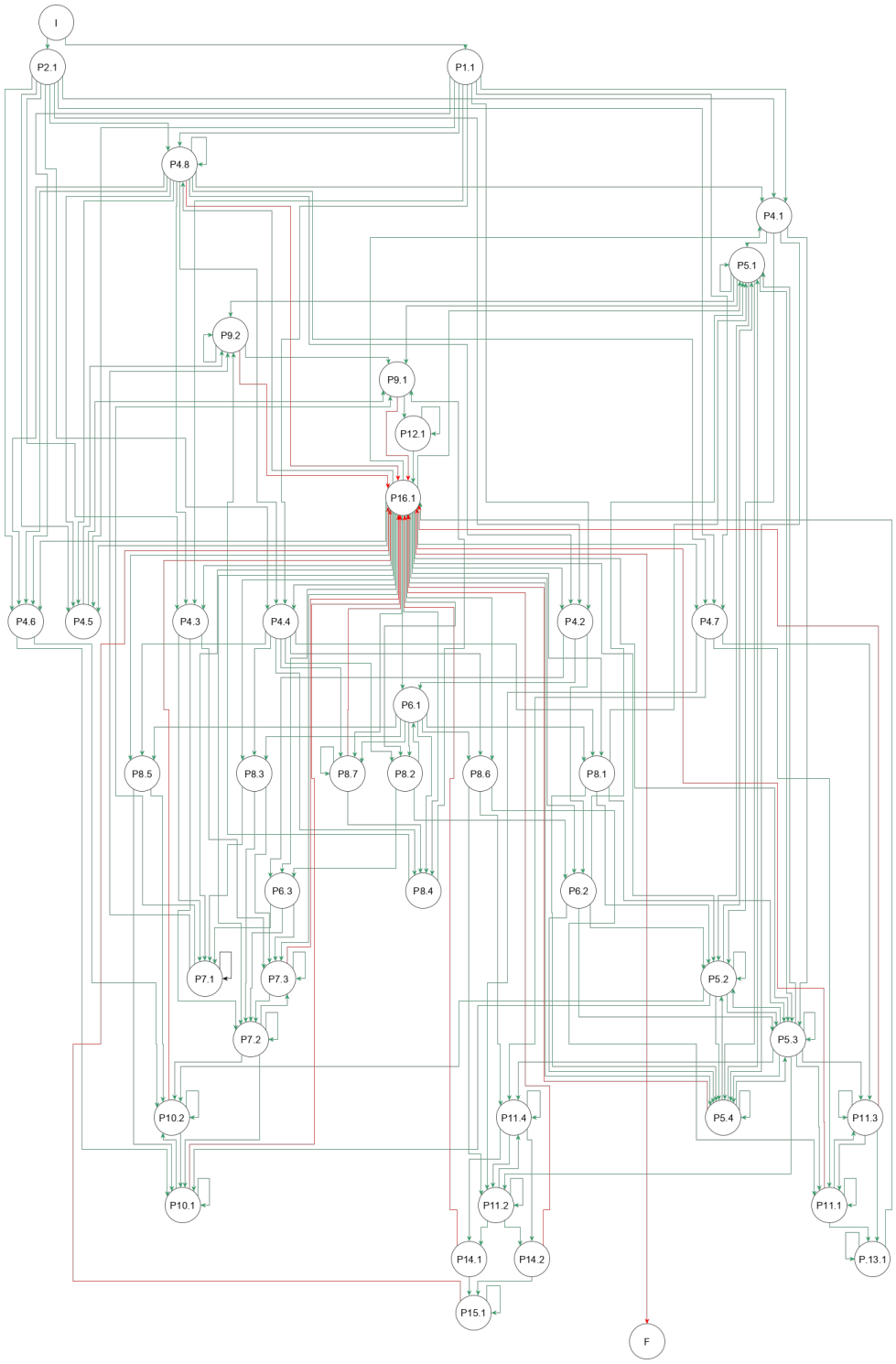
3.1.3 Diagram sterujący

Diagram sterujący nad zdefiniowanymi produkcjami został stworzony w ten sposób, aby generowanie drzewa jak najczęściej odbywało się „w głąb". Stąd jeśli zastosowano na przykład produkcję 4.1, gdzie etykieta $v(3) = DE$, to następnymi możliwymi produkcjami do zastosowania są produkcje 5.1, 5.2, 5.3 i 5.4, gdzie $v(1) = DE$.

Produkcja 16.1 jest produkcją identycznosciową, która została zdefiniowana po to, aby generowane drzewo atrybutowane składało się z wszystkich cech, które zostały przekazane do wierzchołka n_I diagramu sterującego zasilając graf startowy. Wierzchołek ją reprezentujący znajduje się środkowej części grafu przedstawionego na rysunku 3.1. Tylko z niego można osiągnąć wszystkie produkcje, za wyjątkiem produkcji 1.1 i 2.1, których etykieta $v(1)$ w lewej stronie produkcji jest różna od $NF2NF$, $CF2NF$, $TF2NF$. Dodatkowo tylko po nieudanej próbie zastosowania wszystkich produkcji, które możliwe są do osiągnięcia z wierzchołka $P16.1$ można osiągnąć wierzchołek n_F.

	i	P1.1	P2.1	P4.1	P4.2	P4.3	P4.4	P4.5	P4.6	P4.7	P4.8	P5.1	P5.2	P5.3	P5.4	P6.1	P6.2	P6.3	P7.1	P7.2	P7.3	P8.1	P8.2	P8.3	P8.4	P8.5	P8.6	P8.7	P9.1	P9.2	P10.1	P10.2	P11.1	P11.2	P11.3	P11.4	P12.1	P13.1	P14.1	P14.2	P15.1	P16.1	F
i		Y	Y																																									
P1.1				Y	Y	Y	Y	Y	Y	Y	Y																																	
P2.1				Y	Y	Y	Y	Y	Y	Y	Y																																	
P4.1												Y	Y	Y	Y																													
P4.2																Y	Y	Y																										
P4.3																			Y	Y	Y																							
P4.4																						Y	Y	Y	Y	Y	Y	Y															
P4.5																													Y	Y													
P4.6																															Y	Y											
P4.7																																	Y	Y	Y	Y							
P4.8				Y	Y	Y	Y	Y	Y	Y	Y																																N	
P5.1												Y	Y	Y	Y														Y	Y														
P5.2												Y	Y	Y	Y																Y	Y												
P5.3												Y	Y	Y	Y																		Y	Y	Y	Y								
P5.4												Y	Y	Y	Y																												N	
P6.1																						Y	Y	Y	Y	Y	Y	Y															
P6.2												Y	Y	Y	Y																													
P6.3																			Y	Y	Y																							
P7.1																			Y										Y	Y														
P7.2																				Y	Y										Y	Y												
P7.3																				Y	Y																						N	
P8.1												Y	Y	Y	Y																													
P8.2																Y	Y	Y																										
P8.3																			Y	Y	Y																							
P8.4																													Y	Y													
P8.5																															Y	Y											
P8.6																																	Y	Y	Y	Y							
P8.7																									Y			Y															N	
P9.1																																					Y						N	
P9.2																													Y	Y													N	
P10.1																															Y	Y											N	
P10.2																															Y	Y											N	
P11.1																																	Y		Y			Y					N	
P11.2																																		Y		Y				Y	Y		N	
P11.3																																	Y		Y			Y					N	
P11.4																																		Y		Y				Y	Y		N	
P12.1																																					Y						Y	
P13.1																																						Y					Y	
P14.1																																									Y		N	
P14.2																																									Y		N	
P15.1																																									Y		N	
P16.1				Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y															N
F																																												

Tabela 3.3: Diagram sterujący w postaci macierzy sąsiedztwa nad zdefiniowanymi produkcjami. Źródło: opracowanie własne



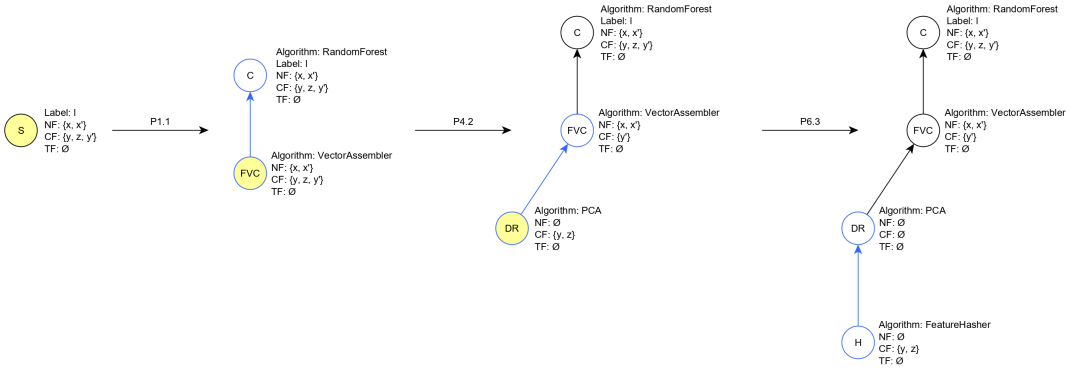
Rysunek 3.1: Diagram sterujący w postaci grafu nad zdefiniowanymi produkcjami na podstawie 3.1.3. Zieloną krawędzią oznaczone są krawędzie Y, krawędzią czerwoną krawędzie N. Źródło: opracowanie własne

3.1.4 Przykładowe wyprowadzenie

Niech $G = (V, W, A, B, P, S, C)$ i V, W, A, B takie jak w tabeli 3.1, P takie jak zdefiniowane w podrozdziale 3.1.2.3, S to atrybutowany graf składający się z jednego wierzchołka S oraz digram sterujący CD z 3.1.

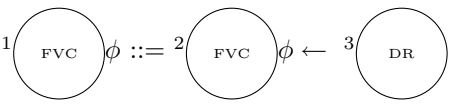
Załóżmy, że w zbiorze danych wejściowych są dwie cechy numeryczne ($\{x, x'\}$), trzy cechy kategorialne ($\{y, y', z\}$) i nie ma żadnej cechy tekstowej, a wartość atrybutu decyzyjnego to: l. Wartości te zostaną przekazane do wierzchołka startowego n_I diagramu sterującego CD, a w następstwie przeniesione do grafu startowego S jako atrybuty $NF = \{x, x'\}$, $CF = \{y, y', z\}$, $TF = \varnothing$ oraz $Label = l$.

Z wierzchołka n_I można osiągnąć wierzchołki $P.1$ oraz $P.2$ pod warunkiem, że zastosowanie produkcji z etykiet wierzchołków jest możliwe. To znaczy wynik ewaluacji predykatu stosowalności jest równy *true*. Jak widać na rysunku 3.2 graf startowy został zamieniony grafem otrzymanym w wyniku zastosowania produkcji $P1.1$. Tę produkcję można zastosować zawsze. Wartości atrybutów NF, CF i NF z grafu lewej strony produkcji zostały jednakowo przeniesione do wierzchołka 1 i 2 grafu prawej strony produkcji odpowiednio do atrybutów NF, CF i NF, atrybut *Algorithm* reprezentujące nazwę algorytmu został przypisany w wyniku zastosowania funkcji Ψ_{alg}, której wynikiem dla wierzchołka z etykietą C jest $RandomForest$, a dla FVC jest $VectorAssembler$. Wartość atrybutu *Label* została przepisana do wierzchołka 1 grafu prawej strony produkcji.



Rysunek 3.2: Zastosowanie produkcji 1.1, 4.2, 6.3. Źródło: opracowanie własne

Następnie z wierzchołka $P1.1$ można osiągnąć kolejno wierzchołki $P4.1$, $P4.2$, $P4.3$, $P4.4$, $P4.5$, $P4.7$ oraz $P4.8$ pod warunkiem, że zastosowanie tych produkcji jest możliwe. W rozważanym przykładzie zastosowano produkcję $P4.2$. Jej zastosowanie było możliwe, bo znaleziono w grafie podgraf lewej strony produkcji $P4.2$ i spełniony został predytkat stosowalności. W wyniku zastosowania produkcji generycznej wierzchołek 2 w grafie, w którym graf z prawej strony produkcji został osadzony, nadal jest połączony skierowaną krawędzią z wierzchołkiem 1 tak jak przed zastosowaniem produkcji $P4.2$.

$P4.2 :$ 

$\pi :$ *true*, bo $|NF(1)| = 2, |CF(1)| = 3, |TF(1)| = 0$

F :

$f_{Algorithm}(3) = PCA$, bo $v(3) = DR \wedge \Psi_{alg}(v(3)) = PCA$
$f_{Algorithm}(2) = VectorAssembler$, bo Algorithm(1) $= VectorAssembler$
$f_{NF}(3) = \varnothing$, bo $NF(1) = \{x, x'\} \wedge \varnothing \subset NF(1) \wedge |\varnothing| \geqslant 0$
$f_{CF}(3) = \{y, z\}$, bo $CF(1) = \{y, z, y'\} \wedge \{y, z\} \subset CF(1) \wedge |\{y, z\}| \geqslant 0$
$f_{TF}(3) = \varnothing$, bo $TF(1) = \varnothing$
$f_{NF}(2) = \{x, x'\}$, bo $NF(1) = \{x, x'\} \wedge NF(3) = \varnothing \Rightarrow NF(1) \setminus NF(3) = \{x, x'\}$
$f_{CF}(2) = \{y'\}$, bo $CF(1) = \{y, z, y'\} \wedge CF(3) = \{y, z\} \Rightarrow CF(1) \setminus CF(3) = \{y'\}$
$f_{TF}(2) = \varnothing$, bo $TF(1) = \varnothing \wedge TF(3) = \varnothing \Rightarrow TF(1) \setminus TF(3) = \varnothing$

Następnie z wierzchołka $P4.2$ można osiągnąć kolejno wierzchołki $P6.1$, $P6.2$ oraz $P6.3$, pod warunkiem, że zastosowanie którejś z produkcji 6.1, 6.2 czy 6.3 jest możliwe. W rozważanym przykładzie zastosowano produkcję 6.3.

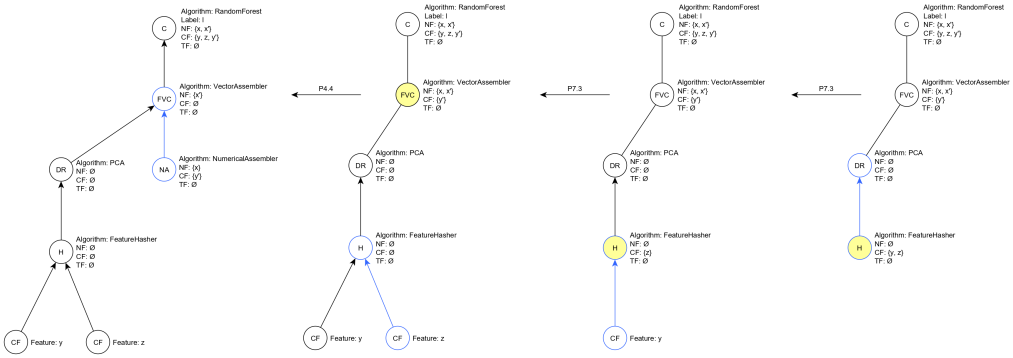
$P6.3$: 

π : $true$, bo $|NF(1)| = 0, |CF(1)| = 2$

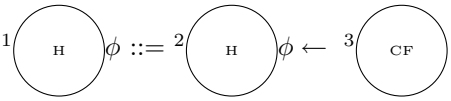
F :

$f_{Algorithm}(3) = FeatureHasher$, bo $v(3) = H \wedge \Psi_{alg}(v(3)) = FeatureHasher$
$f_{Algorithm}(2) = PCA$, bo Algorithm(1) $= PCA$
$f_{NF}(3) = \varnothing$, bo $NF(1) = \varnothing \Rightarrow |NF(1)| \not> 0$
$f_{CF}(3) = \{y, z\}$, bo $CF(1) = \{y, z\} \wedge \{y, z\} \subseteq CF(1) \wedge |\{y, z\}| \geqslant 0$
$f_{NF}(2) = \varnothing$, bo $NF(1) = \varnothing \wedge NF(3) = \varnothing \Rightarrow NF(1) \setminus NF(3) = \varnothing$
$f_{CF}(2) = \varnothing$, bo $CF(1) = \{y, z\} \wedge CF(3) = \{y, z\} \Rightarrow CF(1) \setminus CF(3) = \varnothing$
$f_{TF}(2) = \varnothing$, bo $TF(1) = \varnothing$

Następnie z wierzchołka $P6.3$ można osiągnąć kolejno wierzchołki $P7.1$, $P7.2$ oraz $P7.3$, pod warunkiem, że zastosowanie którejś z produkcji 7.1, 7.2 czy 7.3 jest możliwe. W rozważanym przykładzie zastosowano produkcję P7.3.



Rysunek 3.3: Zastosowanie produkcji 7.3, 7.3, 16.1, 4.4. Źródło: opracowanie własne

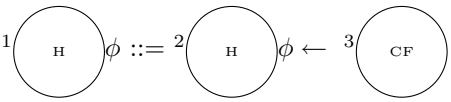
$P7.3$: 

$\pi : true$, bo $|CF(1)| = 2$

F :

$f_{Algorithm}(2) = FeatureHasher$, bo Algorithm(1) $= FeatureHasher$
$f_{Feature}(3) = \{y\}$, bo $CF(1) = \{y, z\} \wedge \{y\} \subset CF(1) \wedge |\{y\}| = 1$
$f_{NF}(2) = \varnothing$, bo $NF(1) = \varnothing$
$f_{CF}(2) = \{z\}$, bo $CF(1) = \{y, z\} \wedge Feature(3) = \{y\} \Rightarrow CF(1) \setminus Feature(3) = \{z\}$

Następnie z wierzchołka $P7.3$ można osiągnąć kolejno wierzchołki $P7.2$, $P7.3$ i $P16.1$. $P7.2$, $P7.3$ pod warunkiem, że zastosowanie którejś z produkcji 7.2, 7.3 jest możliwe. Jeżeli nie, osiągany jest wierzchołek $P16.1$. W rozważanym przykładzie ponownie zastosowano produkcję 7.3.

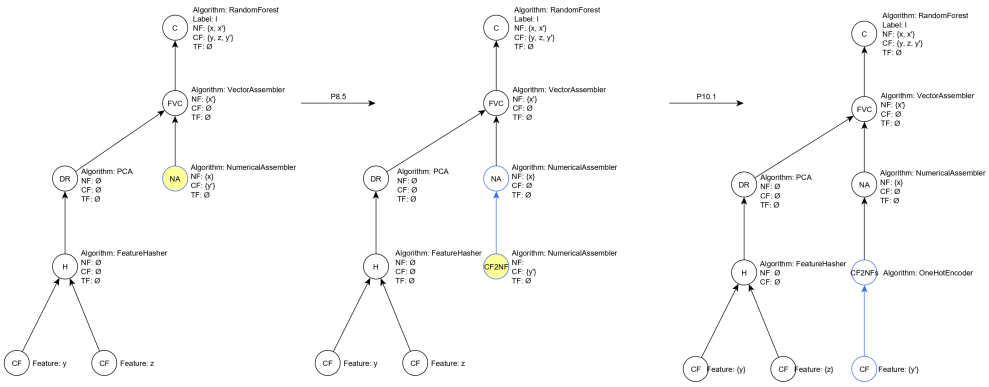
$P7.3$: 

$\pi : true$, bo $|CF(1)| = 1$

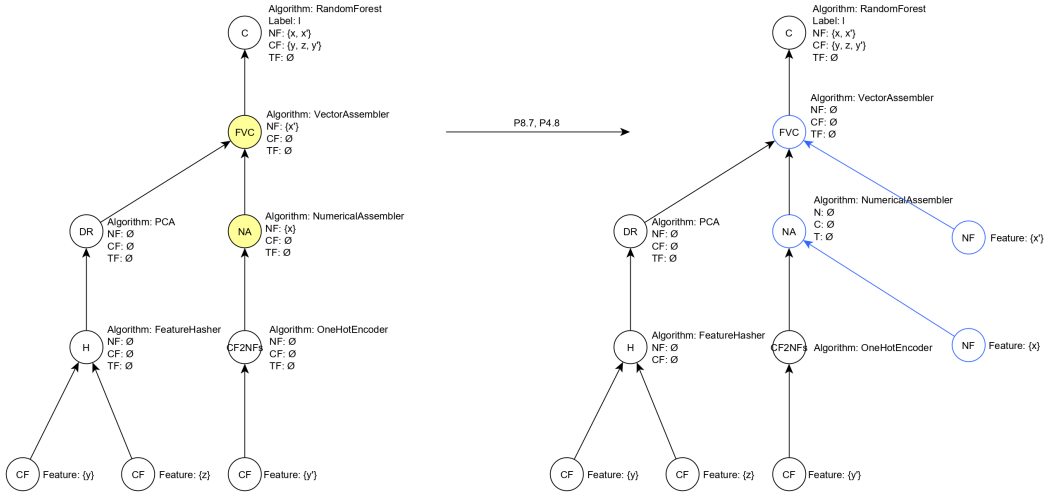
F :

$f_{Algorithm}(2) = FeatureHasher$, bo Algorithm(1) $= FeatureHasher$
$f_{Feature}(3) = \{z\}$, bo $CF(1) = \{z\} \wedge \{z\} \subset CF(1) \wedge |\{z\}| = 1$
$f_{NF}(2) = \varnothing$, bo $NF(1) = \varnothing$
$f_{CF}(2) = \varnothing$, bo $CF(1) = \{z\} \wedge Feature(3) = \{z\} \Rightarrow CF(1) \setminus Feature(3) = \{z\}$

Po zastosowaniu produkcji 7.3 aktualnie osiągnięty wierzchołek diagramu sterującego to $P7.3$. Wierzchołek $P7.2$ oraz $P7.3$ nie może zostać osiągnięty, ponieważ wynikiem testu predykatu stosowalności dla produkcji 7.2 oraz 7.3 ($\pi : |CF(1)| > 0$) jest $false$. Innymi słowy w grafie nie istnieje taki podgraf, który mógłby zostać zastąpiony prawą stroną z danej produkcji, ze względu na wynik predykatu stosowalności. Zatem osiągany jest wierzchołek $P16.1$. Stosowanie kolejnych produkcji z zastosowaniem diagramu sterującego odbywa się analogicznie.

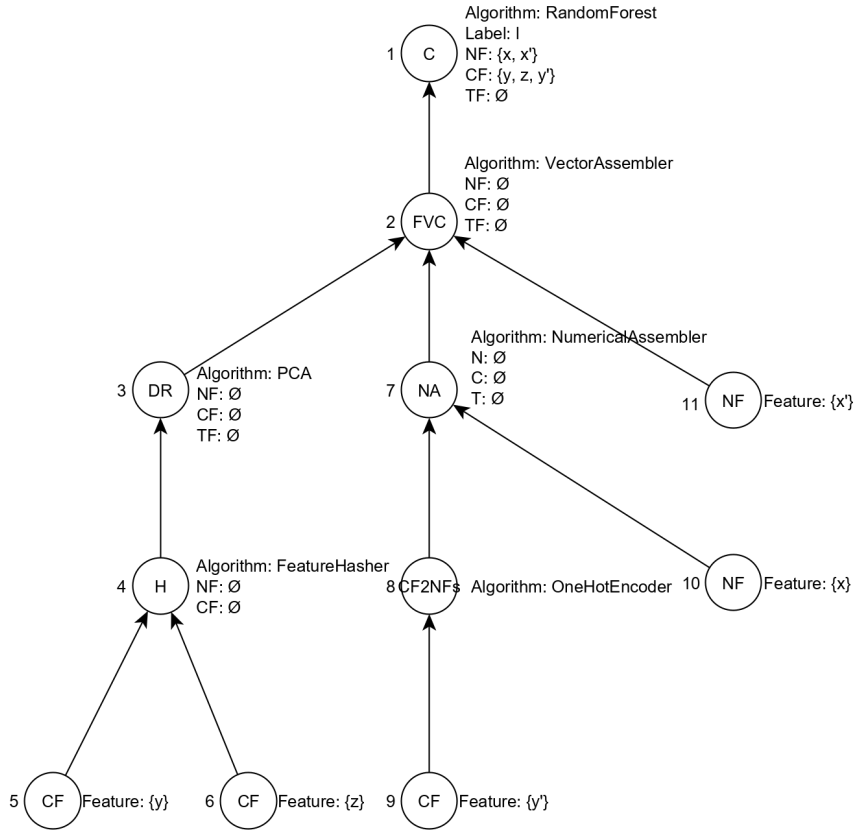


Rysunek 3.4: Zastosowanie produkcji 8.5, 10.1. Źródło: opracowanie własne

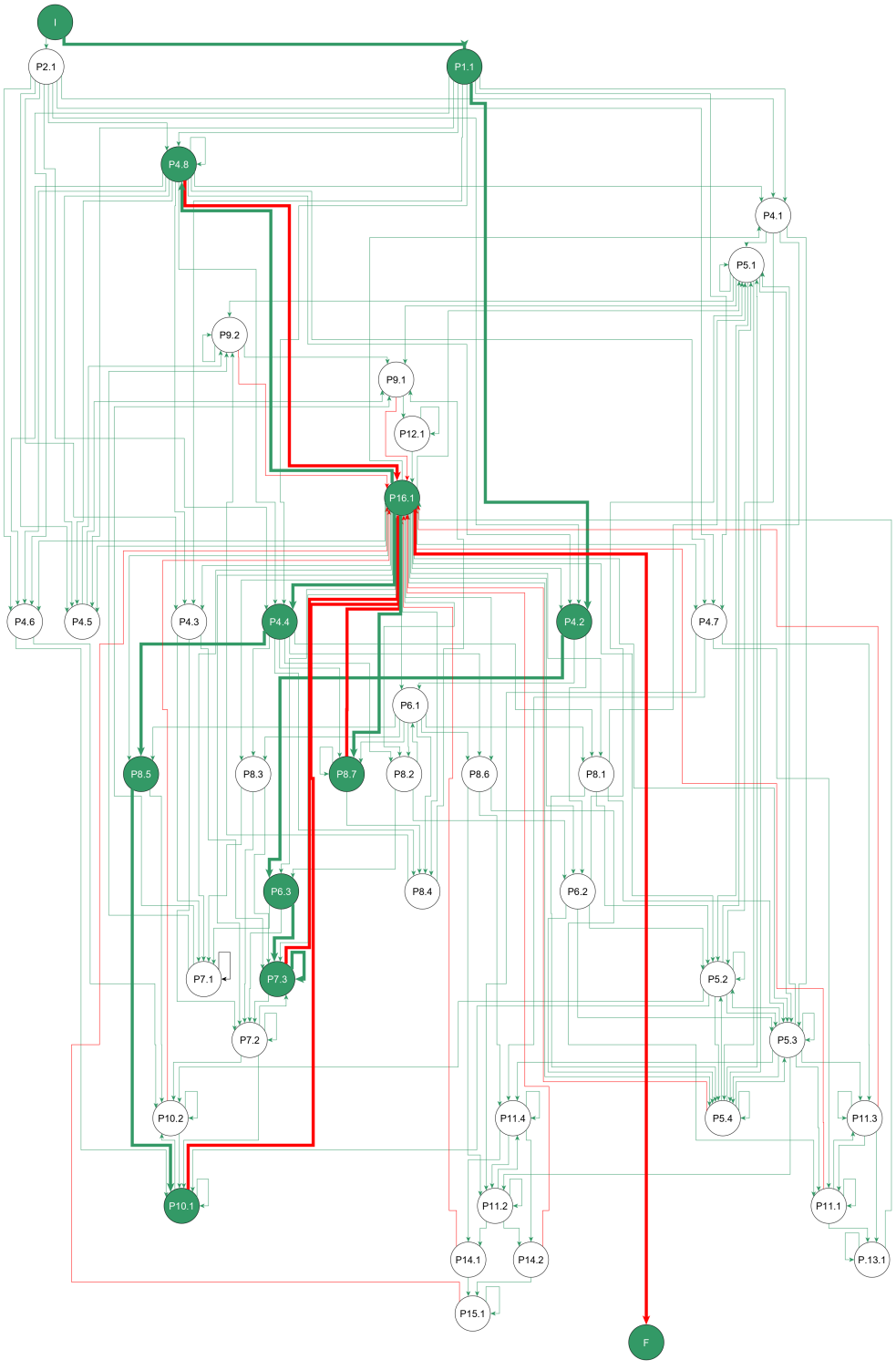


Rysunek 3.5: Zastosowanie produkcji 8.7, 4.8. Źródło: opracowanie własne

Po zastosowaniu ostatniej produkcji 4.8 z rysunku 3.5 jedynym możliwym wierzchołkiem diagramu sterującego do osiągnięcia jest wierzchołek $P16.1$, ponieważ żadna z produkcji 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 i 4.8 nie może zostać zastosowana ze względu na to, że wartości atrybutów NF, CF, TF przy wierzchołku o etykiecie FVC są równe $\varnothing$. Osiągając wierzchołek $P16.1$ nie można zastosować żadnej produkcji, która byłaby możliwa do zastosowania. Dlatego też n_F jest możliwy do osiągnięcia. Przechodząc do wierzchołka o etykiecie F jesteśmy w wierzchołku końcowym, a zatem został wygenerowany poprawny graf atrybutowany. Jest to drzewo reprezentujące model klasyfikujący składający się z klas przekształceń oraz cech niezbędnych do zbudowania modelu klasyfikującego.



Rysunek 3.6: Przykładowe drzewo będące efektem zastosowania ciągu produkcji 1.1, 4.2, 6.3, 7.3, 7.3, 4.4, 8.5, 10.1 i przejścia wierzchołków diagramu sterującego w kolejności: I, 1.1, 4.2, 6.3, 7.3, 7.3, 16.1, 4.4, 8.5, 10.1, 16.1, F. Źródło: opracowanie własne



Rysunek 3.7: Przejście diagramu sterującego od wierzchołka początkowego n_I do wierzchołka końcowego n_F z zastosowaniem poszczególnych produkcji dla przedstawionego przykładu. Zieloną krawędzią oznaczone są krawędzie Y, krawędzią czerwoną krawędzie N. Źródło: opracowanie własne

3.2 Implementacja programowania genetycznego

Zaproponowana implementacja programowania genetycznego jest implementacją programowania genetycznego z wykorzystaniem gramatyki formalnej [Whi95] – atrybutowej programowalnej gramatyce grafowej zdefiniowanej w rozdziale 3.1.

3.2.1 Zbiór terminalny i funkcyjny

W celu tworzenia populacji modeli klasyfikujących definiuje się zbiór funkcyjny F_S (def. 2.2.1) oraz zbiór terminalny T_S (def. 2.2.1). Zbiór F_S zawiera wartości komponentów klas algorytmów oraz transformacji wartości cech. Zbiór T_S zawiera wartości komponentów reprezentujących cechy w zbiorach danych treningowych i testowych. Wartości tych zbiorów definiuje się i interpretuje się jako poszczególne wartości zdefiniowanego zbioru alfabetu etykiet wierzchołkowych V (def. 3.1):

- $T_S = \{NF, CF, TF\}$, gdzie $\{NF, CF, TF\} \subset V$

Zbiór terminalny	
Komponent	*Wartości*
Cech w zbiorach danych	NF, CF, TF

Tabela 3.4: Zbiór terminalny, jego komponenty i wartości. Źródło: opracowanie własne

- $F_S = V \setminus (\{S\} \cup T_S)$

Zbiór funkcyjny	
Komponent	*Wartości*
Klas algorytmów	C, ENS_C, FVC, DE, DR, H, NA, TA, TA_S
Klas transformacji wartości cech	$NF2NF$, $NF2NF_S$, $CF2NF$, $CF2NF_S$, $TF2NF$, $TF2NF_S$, $TF2TF$

Tabela 3.5: Zbiór funkcyjny, jego komponenty i wartości. Źródło: opracowanie własne

3.2.2 Repozytorium hiperparametrów

Zbiór atrybutów wierzchołkowych A (def. 3.1) zawiera ogólne atrybuty w kontekście opisu modelu klasyfikującego. Nie wyróżnia się atrybutów wierzchołkowych, które reprezentowałyby odpowiednie hiperparametry dla algorytmów reprezentowanych przez wartości atrybutów $Algorithm$ i $Classifiers$, ponieważ te są różne w kontekście różnych algorytmów. Dla algorytmu lasu losowego reprezentowanego przez wartość atrybutu wierzchołkowego $RandomForest$ hiperparametrami będą na przykład: maksymalna wysokość drzew w lesie czy miara czystości (ang. impurity) jego węzłów. Dla algorytmu perceptronu wielowarstwowego reprezentowanego przez wartość atrybutu wierzchołkowego $MultilayerPerceptron$ hiperparamaterem będzie na przykład liczba warstw ukrytych sieci i liczba

neuronów w każdej warstwie. Jak można zauważyć, możliwe wartości hiperparametrów w kontekście rozważanych algorytmów diametralnie się różnią. Bez zdefiniowania dodatkowej struktury, która uwzględnia możliwe wartości hiperparametrów można by założyć, że stworzony program na bazie drzewa atrybutowanego będzie przyjmować pewne domyślne wartości hiperparametrów dla poszczególnych algorytmów. Jednakże takie założenie powoduje, że informacja jest domniemana i nieustrukturyzowana, a zatem nie można do niej zastosować operatorów genetycznych – na przykład operatora mutacji, który, przykładowo, zmieniłby liczbę warstw ukrytych w algorytmie perceptronu wielowarstwowego. Dlatego też definiuje się pomocniczą strukturę nazwaną jako repozytorium hiperparametrów.

Repozytorium hiperparametrów R_H to uporządkowana lista list hiperparametrów h, gdzie:

- H_{IS} to skończony zbiór strategii integracji klasyfikatorów, gdzie H_{IS} = {majority voting, weighted majority voting, naive bayes integration, kullbeck leibler integration, pca integration, single layered neural network integration},
- H_{NB} to skończony zbiór typów modeli algorytmu naiwnego klasyfikatora bayesowskiego, gdzie H_{NB} = {bernoulli, multinomial},
- H_{TI} to skończony zbiór miar czystości węzłów drzew w algorytmach wykorzystujących drzewa decyzyjne, gdzie H_{TI} = {entropy, gini},
- H_{TS} to skończony zbiór strategii podziału obszaru danych w algorytmach wykorzystujących drzewa decyzyjne, gdzie H_{TS} = {all, one third, sqrt, $\log_2$}
- H_{BP} to skończony zbiór algorytmów wstecznej propagacji błędów w algorytmie perceptronu wielowarstwowego, gdzie H_{BP} = {gradient descent, l-bgfs},
- H_C to skończony zbiór strategii wyboru istotnych deskryptorów numerycznych cech w algorytmie bazującym na statystycznym teście niezależności chi-kwadrat, gdzie H_C = {top, percentile, false positive rate, false discovery rate, family wise error rate},
- H_I to skończony zbiór strategii imputacji danych, gdzie H_I = {mean, median},
- $\mathbb{R}$ to zbiór liczb rzeczywistych,
- $\mathbb{C}$ to zbiór liczb całkowitych,
- $\mathbb{N}$ to zbiór liczb naturalnych łącznie z zerem,
- $\mathbb{B}$ to skończony zbiór wartości logicznych logiki dwuwartościowej, gdzie $\mathbb{B}$ = {true, false},
- $h = (h_1, ..., h_n)$, to n elementowa uporządkowana lista wartości hiperparametrów danego algorytmu. Kolejnymi wartościami listy mogą być wartości wymienionych wyżej zbiorów lub uporządkowane listy k elementowe, których elementy mogą należeć do wyżej wymienionych zbiorów. $h = ()$ oznacz pustą listę.

Klasy algorytmów	Algorytmy	Listy hiperparametrów h i dziedziny ich elementów
ENS_C	Stacking Ensemble	h = ($h_1 \in H_{IS}$, h_2, ... h_k to listy hiperparametrów zdefiniowanych dla klasy algortymów C)
C	Naive Bayes	h = ($h_1 \in H_{NB}$, $h_2 \in \mathbb{B}$), $h_2 \in \mathbb{R}$
	Logistic Regression	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{R}$, $h_3 \in \mathbb{N}$), $h_2 \geqslant 0$, $h_3 \geqslant 2$
	Decision Tree, Stochastic Gradient Boosted Decision Trees, Random Forest	h = ($h_1 \in H_{TI}$, $h_2 \in H_{TS}$, $h_3 \in \mathbb{N}$, $h_4 \in \mathbb{R}$, $h_5 \in \mathbb{N}$), $h_3, h_4 \geqslant 0$, $h_5 \geqslant 1$
	Multilayer Perceptron	h = ($h_1 \in H_{BP}$, h_2 to uporządkowana lista k elementowa taka, że jej wartości należą do $\mathbb{N}$, $h_3 \in \mathbb{R}$), $h_3 > 0$, $k > 0$
	Linear Support Vector Machine	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{R}$, $h_3 \in \mathbb{N}$), $h_2 \geqslant 0$, $h_2 \geqslant 2$
FVC	Chi-Squared Based Selector	h = ($h_1 \in H_C$, $h_2 \in \mathbb{N}$, $h_3 \in \mathbb{R}$, $h_4 \in \mathbb{R}$, $h_5 \in \mathbb{R}$), $h_1 > 3$, $h_2, h_3, h_4, h_5 \in [0, 1]$
DR	PCA	h = ($h_1 \in \mathbb{N}$), $h_1 > 0$
DE	Polynomial Expansion	h = ($h_1 \in \mathbb{N}$), $h_1 \geqslant 1$
	Murmur Hash	h = ($h_1 \in \mathbb{N}$), $h_1 > 0$
$NF2NF_{(S)}$	Standard Scaler	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{B}$)
	Normalizer	h = ($h_1 \in \mathbb{N}$), $h_1 \geqslant 2$
	Min Max Scaler	h = ($h_1 \in \mathbb{R}$, $h_2 \in \mathbb{R}$)
	Bucketizer	h = (h_1 to uporządkowana lista k elementowa taka, że jej wartości należą do $\mathbb{R}$), $k > 0$
	Binarizer	h = ($h_1 \in \mathbb{R}$), $h_1 \geqslant 0$
	Imputer	h = ($h_1 \in H_I$, $h_2 \in \mathbb{R}$)
	IDF	h = ($h_1 \in \mathbb{N}$), $h_1 \geqslant 0$
$CF2NF_{(S)}$	Tokenizer	h = ($h_1 \in \mathbb{N}$), $h_1 > 0$
	Count Vectorizer	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{R}$, $h_3 \in \mathbb{R}$, $h_4 \in \mathbb{R}$, $h_5 \in \mathbb{R}$), $h_2, h_3, h_4, h_5 \geqslant 1$
$TF2NF_{(S)}$	HTF	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{N}$), $h_2 > 0$
	HTF - IDF	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{N}$, $h_3 \in \mathbb{N}$), $h_2 \geqslant 0$, $h_3 > 0$
	Count Vectorizer - IDF	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{R}$, $h_3 \in \mathbb{R}$, $h_4 \in \mathbb{R}$, $h_5 \in \mathbb{R}$, $h_6 \in \mathbb{N}$), $h_2, h_3, h_4, h_5, h_6 \geqslant 1$
	Word2Vec	h = ($h_1 \in \mathbb{N}$, $h_2 \in \mathbb{N}$, $h_3 \in \mathbb{N}$, $h_4 \in \mathbb{N}$, $h_5 \in \mathbb{N}$), $h_1 > 0$, $h_2 \geqslant 0$, $h_3, h_4, h_5 > 0$
	Count Vectorizer	h = ($h_1 \in \mathbb{B}$, $h_2 \in \mathbb{R}$, $h_3 \in \mathbb{R}$, $h_4 \in \mathbb{R}$, $h_5 \in \mathbb{R}$), $h_2, h_3, h_4, h_5 \geqslant 1$
TF2TF	nGram	h = ($h_1 \in \mathbb{N}$), $h_1 \geqslant 1$

Tabela 3.6: Dziedziny elementów list hiperparametrów dla poszczególnych algorytmów. Uwzględniono tylko te listy, które nie są puste. Źródło: opracowanie własne

Klasy algorytmów	Algorytmy	Listy hiperparametrów h i dziedziny ich elementów
C	Naive Bayes	h = (multinomial, 1)
	Logistic Regression	h = (true, 0, 2)
	Decision Tree, Stochastic Gradient Boosted Decision Trees, Random Forest	h = (entropy, all, 5, 0, 1)
	Multilayer Perceptron	h = (l-bfgs, (10, 5, 10), 0.03)
	Linear Support Vector Machine	h = (true, 0, 2)
FVC	Vector Assembler	h = ()
	Chi-Squared Based Selector	h = (top, 50, 0.1, 0.05, 0.05, 0.05)
NA	Numerical Assembler	h = ()
TA, TA_S	Text Assembler	h = ()
DR	PCA	h = (3)
DE	Polynomial Expansion	h = (2)
	Murmur Hash	h = (2^6)
NF2NF, $NF2NF_S$	Standard Scaler	h = (false, true)
	Normalizer	h = (2)
	Min Max Scaler	h = (0, 1)
	Max Abs Scaler	h = ()
	Bucketizer	h = $((-2^{63}+1, -0.5, 0, 0.5, 2^{63}-1))$
	Binarizer	h = (0)
	Imputer	h = (mean, 0)
	IDF	h = (0)
CF2NF, $CF2NF_S$	One Hot Encoder	h = ()
	String Indexer	h = ()
	Tokenizer	h = (1)
	Count Vectorizer	h = (false, 1, $2^{63}-1$, 1, 2^{18})
TF2NF, $TF2NF_S$	HTF	h = (false, 2^{18})
	HTF - IDF	h = (false, 0, 2^{18})
	Count Vectorizer - IDF	h = (false, 1, $2^{63}-1$, 1, 2^{18}, 0)
	Word2Vec	h = (1000, 5, 5, 100, 1)
	Count Vectorizer	h = (false, 1, $2^{63}-1$, 1, 2^{18})
TF2TF	nGram	h = (2)
	Normalizer	h = ()
	Stemmer	h = ()
	Lemmatizer	h = ()

Tabela 3.7: Domyślne wartości elementów list hiperparametrów dla poszczególnych algorytmów. Źródło: opracowanie własne na postawie domyślnych wartości w bibliotece Apache Spark MLlib

Dodatkowo, definiuje się funkcję Ω przypisującą wierzchołkowi drzewa domyślną listę hiperpa-

rametrów dla wartości atrybutu wierzchołkowego reprezentującego nazwę algorytmu. Ω to funkcja $\Omega : N \rightarrow H$, gdzie:

- N to skończony zbiór etykietowanych wierzchołków grafu atrybutowanego,
- H to skończony zbiór możliwych list hiperparametrów.

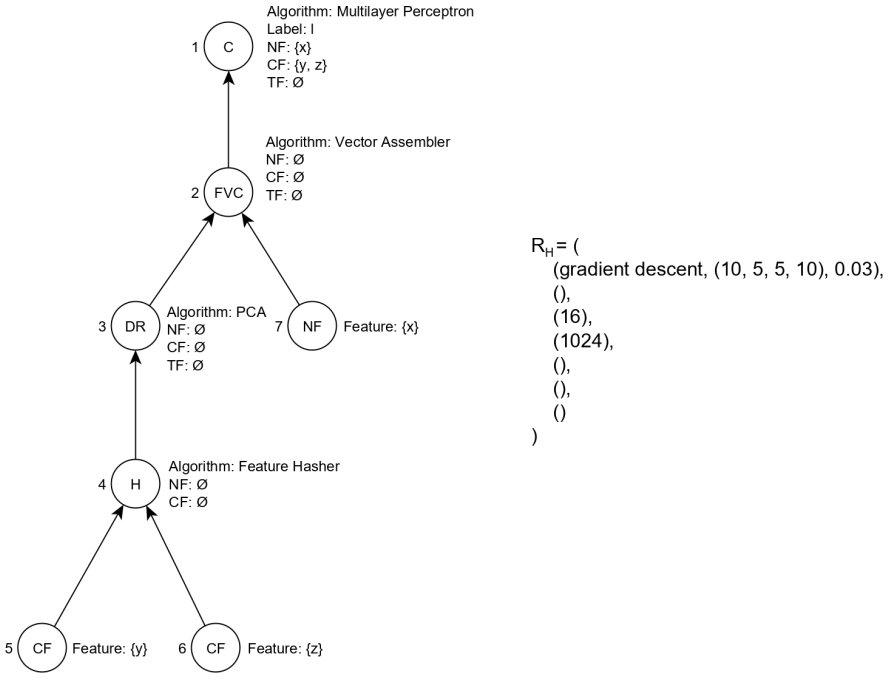
Wartościami zbioru H są listy hiperparametrów zdefiniowane w tabeli 3.6. Przykładowo, jeżeli wierzchołek n nie posiada atrybutu wierzchołkowego $Classifiers$ i posiada atrybut wierzchołkowy *Algorithm*, którego przykładową wartością jest $Multilayer\ Perceptron$, to dla tego wierzchołka n przykładową wartością funkcji $\Omega(n)$ jest (l-bfgs, (10, 5, 10), 0.03). Jeżeli wierzchołek n posiada atrybut wierzchołkowy $Classifiers$, którego przykładowa wartość to zbiór dwuelementowy zawierający wartości *Logistic Regression* i $Multilayer\ Perceptron$ oraz posiada atrybut wierzchołkowy *Algorithm*, którego przykładowa wartość to *Stacking Ensemble*, to dla takiego wierzchołka n przykładową wartością funkcji $\Omega(n)$ jest (majority voting, (true, 0, 2), (l-bfgs, (10, 5, 10), 0.03)). Jeżeli wierzchołek n nie posiada ani atrybutu wierzchołkowego $Algoritm$, ani $Classifiers$, to wtedy wartość funkcji $\Omega(n)$ to (), czyli pusta lista. Dodatkowo $R_H(n)$ oznacza listę hiperparametrów dla wierzchołka n z repozytorium hiperparametrów.

3.2.3 Reprezentacja

Genotyp modelu klasyfikującego składa się z dwóch chromosomów:

- pierwszy chromosom to drzewo atrybutowane jako słowo języka gramatyki grafowej zdefiniowanej w rozdziale 3.1,
- drugi chromosom to lista list zdefiniowana jako repozytorium hiperparametrów (def. 3.2.2) dla drzewa atrybutowanego w ten sposób, że kolejne listy hiperparametrów odpowiadają danemu wierzchołkowi w drzewie z pierwszego chromosomu.

Na rysunku 3.8 przedstawiono przykładowy genotyp. W lewej części rysunku znajduje się drzewo atrybutowane reprezentujące model klasyfikujący dla zbioru danych o atrybucie decyzyjnym l oraz jednej cesze numerycznej ($\{x\}$) i dwóch katagorialnych ($\{y, z\}$). W prawej części rysunku znajduje się repozytorium hiperparametrów, które zawiera siedem list hiperparametrów. Pierwszy element repozytorium $h_1 = (gradient\ descent, (10, 5, 5, 10), 0.03)$ odpowiada wierzchołkowi pierwszemu, którego wartość atrybutu *Algorithm* to $Multilayer\ Perceptron$. Drugi element repozytorium $h_2 = ()$ odpowiada wierzchołkowi drugiemu, którego wartość atrybutu *Algorithm* to $Vector\ Assembler$. Trzeci element repozytorium $h_3 = (16)$ odpowiada wierzchołkowi trzeciemu, którego wartość atrybutu *Algorithm* to PCA. Czwarty element repozytorium $h_4 = (1024)$ odpowiada wierzchołkowi czwartemu, którego wartość atrybutu *Algorithm* to $Feature\ Hasher$. Piąty $h_5 = ()$, szósty $h_6 = ()$ i siódmy element repozytorium $h_7 = ()$ odpowiada wierzchołkowi piątemu, szóstemu i siódmemu, który nie posiada ani atrybutu *Algorithm*, ani atrybutu $Classifiers$. Innymi słowy $R_H = (\Omega(n_1), \Omega(n_2), \Omega(n_3), \Omega(n_4), \Omega(n_5), \Omega(n_6), \Omega(n_7))$. Przykładowo $R_H(n_2) = ()$.



Rysunek 3.8: Genotyp składający się z dwóch przykładowych chromosomów – drzewa atrybutowanego będącego słowem języka zdefiniowanej gramatyki oraz jego repozytorium hiperparametrów. Źródło: opracowanie własne

3.2.4 Funkcja dopasowania

Funkcję dopasowania (def. 2.2.2) do problemu optymalizacji modelu klasyfikującego definiuje się jako funkcję, która dla każdego osobnika przypisze wartość wybranej miary jakości modelu określonej w konfiguracji programowania genetycznego. Miara jakości uzyskiwana jest w ten sposób, że każdy model klasyfikujący stworzony na podstawie genotypu osobnika jest trenowany przy pomocy zbioru treningowego oraz weryfikowany przy pomocy zbioru testowego zdefiniowanego w konfiguracji programowania genetycznego. Tak uzyskana wartość wybranej miary skuteczności jest wartością funkcji dopasowania danego osobnika.

3.2.5 Operatory genetyczne

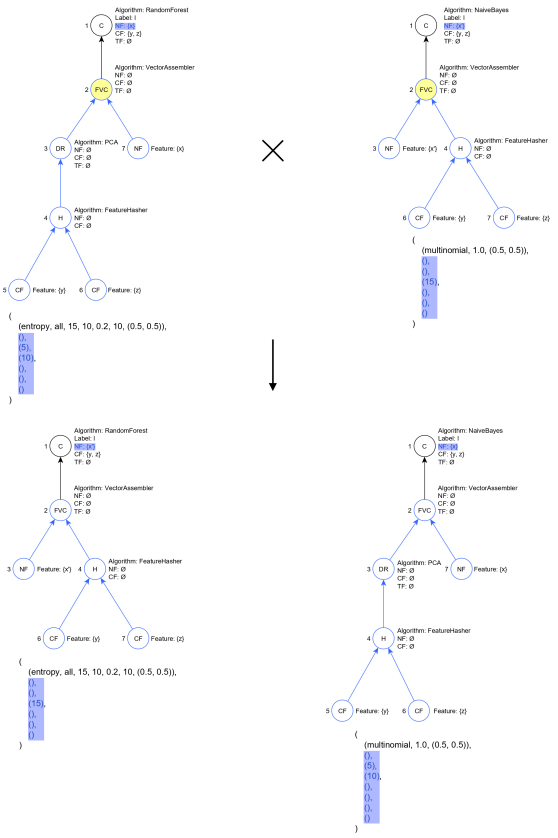
Ze względu na specyfikę zaproponowanego genotypu definiuje się własne operatory genetyczne:

- operatory krzyżowania: poprzez przepinanie poddrzew i hiperparametrów (def. 3.2.5.1), integrowanie drzew w korzeniu o tej samej etykiecie (def. 3.2.5.2),
- operatory mutacji: wartości pojedynczego elementu hiperparametrów (def. 3.2.5.3), wartości atrybutu wierzchołkowego *Algorithm* i odpowiadającej modyfikowanemu wierzchołkowi listy hiperparametrów (def. 3.2.5.4), poddrzewa (def. 3.2.5.5).

3.2.5.1 Operator krzyżowania poprzez przepinanie poddrzew i hiperparametrów

Operator krzyżowania poprzez przepinanie poddrzew i hiperparametrów bazuje na operatorze przepinania poddrzew [Pol01]. Rozszerzenie obejmuje transfer wartości repozytorium hiperparametrów stowarzyszonych z wierzchołkami przepinanych poddrzew. Przebieg zastosowania tego operatora jest następujący:

1. Dla dwojga rodziców znajdź wierzchołek o takiej samej etykiecie pomijając wierzchołki etykietowane etykietą C, ENS_C, NF, CF, TF. Znalezione wierzchołki są teraz korzeniami poddrzew pierwszego i drugiego rodzica,
2. Utwórz pierwszego (drugiego) potomka poprzez przepięcie poddrzewa drugiego (pierwszego) rodzica w miejsca poddrzewa pierwszego (drugiego) rodzica i skopiuj do niego repozytorium hiperparametrów pierwszego (drugiego) rodzica:
   (a) Dla nowoutworzonego potomka zamień istniejące listy hiperparametrów przepinanego poddrzewa na te stowarzyszone z wierzchołkami przepiętego poddrzewa,
   (b) Zaktualizuj wartości atrybutów wierzchołkowych korzenia NF, CF i TF na podstawie etykiet liści i ich atrybutów wierzchołkowych $Feature$.

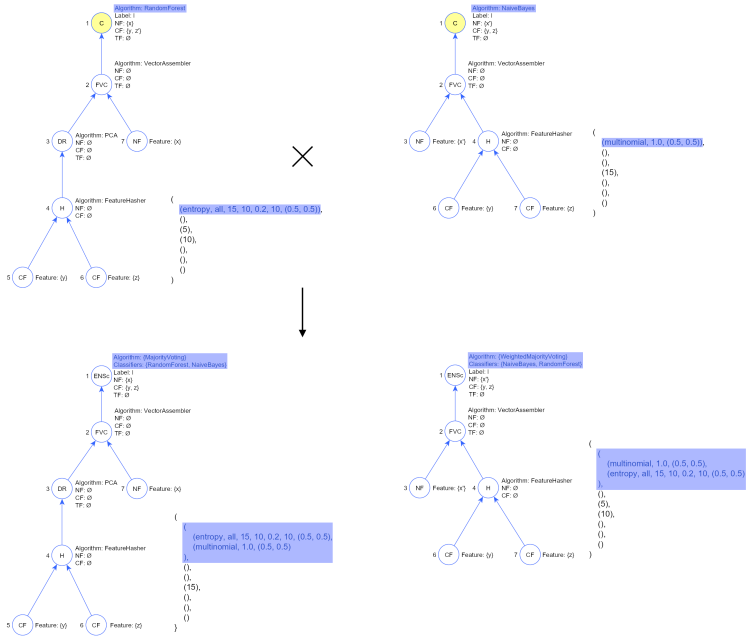


Rysunek 3.9: Przykładowe zastosowanie operatora przepinanie poddrzew i hiperparametrów. Źródło: opracowanie własne

3.2.5.2 Operator krzyżowania poprzez integrowanie drzew w korzeniu o tej samej etykiecie

Operator krzyżowania poprzez integrowanie drzew w korzeniu o tej samej etykiecie pozwala tworzyć osobniki reprezentujące model klasyfikujący z więcej niż jednym algorytmem klasyfikacji odpowiednio je integrując. Operator może być zastosowany tylko dla tych rodziców, których etykieta korzenia to C i wartości atrybutu wierzchołkowego $Feature$ w liściach są w $\frac{2}{3}\%$ takie same. Przez r_1 oznacza się wierzchołek będący korzeniem pierwszego rodzica, przez r_2 wierzchołek będący korzeniem drugiego rodzica. Przebieg zastosowania tego operatora jest następujący:

1. Utwórz pierwszego potomka na podstawie pierwszego i drugiego rodzica tak, że:
   (a) korzeń nowoutworzonego potomka jest zamieniony na wierzchołek ch_1 tak, że nowy wierzchołek jest teraz etykietowany etykietą ENS_C i jego atrybuty wierzchołkowe (zgodnie z def. 3.1.2.1), to $f_{Algorithm}(ch_1) = \Psi(ch_1)$, $f_{Classifiers}(ch_1) = \{Algorithm(r_1), Algorithm(r_2)\}$ i $f_{Label}(ch_1) = Label(r_1)$, $f_{NF}(ch_1) = NF(r_1)$, $f_{CF}(ch_1) = CF(r_1)$, $f_{TF}(ch_1) = TF(r_1)$,
   (b) $R_{H_{r_1}}(ch_1) = (R_{H_{r_1}}(ch_1), R_{H_{r_2}}(ch_2))$ (zgodnie z def. 3.2.2).
2. Utwórz drugiego potomka na podstawie pierwszego i drugiego rodzica tak, że:
   (a) korzeń nowoutworzonego potomka jest zamieniony na wierzchołek ch_2 tak, że nowy wierzchołek jest teraz etykietowany etykietą ENS_C i jego atrybuty wierzchołkowe (zgodnie z def. 3.1.2.1), to $f_{Algorithm}(ch_2) = \Psi(ch_2)$, $f_{Classifiers}(ch_2) = \{Algorithm(r_2), Algorithm(r_1)\}$ i $f_{Label}(ch_2) = Label(r_2)$, $f_{NF}(ch_2) = NF(r_2)$, $f_{CF}(ch_2) = CF(r_2)$, $f_{TF}(ch_2) = TF(r_2)$,
   (b) $R_{H_{r_2}}(ch_2) = (R_{H_{r_2}}(ch_2), R_{H_{r_1}}(ch_1))$ (zgodnie z def. 3.2.2).



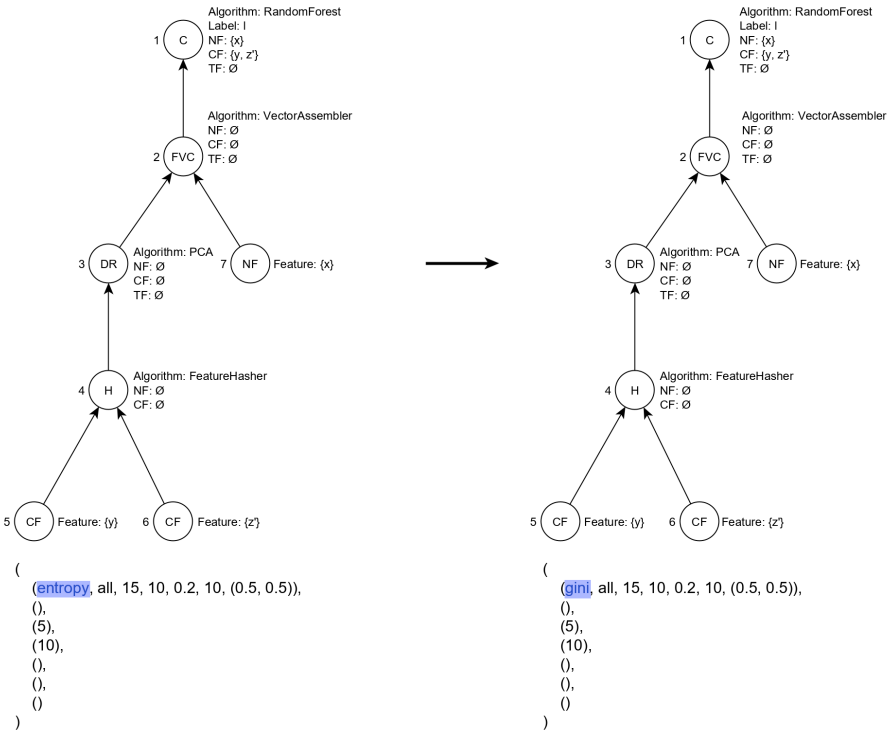
Rysunek 3.10: Przykładowe zastosowanie operatora krzyżowania poprzez integrowanie drzew w korzeniu o tej samej etykiecie. Źródło: opracowanie własne

3.2.5.3 Operator mutacji wartości pojedynczego elementu listy hiperparametrów

Operator mutacji wartości pojedynczego elementu listy hiperparametrów ma zastosowanie do losowo wybranej listy znajdującej się w repozytorium hiperparametrów osobnika. W wylosowanej liście losowo wybiera się jej element e. Jego nową wartość określa zgodnie z dziedziną elementu zdefiniowaną w tabeli 3.6, tak, że:

- jeżeli $e \in \mathbb{R}$, to wtedy $e = e + sgn(\mathcal{U}(0,1) - 0.5) * \frac{(e+1)*U(0,2)}{2}$
- jeżeli $e \in \mathbb{C}$, to wtedy $e = [e + sgn(\mathcal{U}(0,1) - 0.5) * \frac{(e+1)*U(0,2)}{2}]$
- jeżeli $e \in \mathbb{N}$, to wtedy $e = max(0, e + sgn(\mathcal{U}(0,1) - 0.5) * e * \mathcal{U}(1, e+1))$
- jeżeli $e \in \mathbb{B}$, to wtedy $e = inv(e)$, gdzie $inv(e) = \begin{cases} true, & \text{jeżeli } e = false \\ false, & \text{jeżeli } e = true \end{cases}$
- jeżeli $e \in H_{IS} \vee e \in H_{NB} \vee e \in H_{TI} \vee e \in H_{TS} \vee e \in H_{BP} \vee e \in H_C \vee e \in H_I$, to wtedy e to losowym elementem tego zbioru różny od e, do którego należy e.

Gdzie $\mathcal{U}(a,b)$ oznacza liczbę losowo generowaną zgodnie z rozkładem jednorodnym między a i b, gdzie $b > a$, $sgn(x)$ oznacza funkcję signum, $[x]$ oznacza część całkowitą z x.



Rysunek 3.11: Przykładowe zastosowanie operatora mutacja wartości pojedynczego elementu listy hiperparametrów. Źródło: opracowanie własne

3.2.5.4 Operator mutacji wartości atrybutu wierzchołkowego *Algorithm* i odpowiadającej modyfikowanemu wierzchołkowi listy hiperparametrów

Operator mutacji wartości atrybutu wierzchołkowego *Algorithm* i odpowiadającej modyfikowanemu wierzchołkowi listy hiperparametrów ma zastosowanie do losowo wybranego wierzchołka n posiadajacego atrybut *Algorithm* w taki sposób, że $f_{Algorithm}(n) = \Psi(n)$ (zgodnie z def. 3.1.2.1), a nowa wartość listy hiperparametrów $R_H(n)$ to domyślna wartość dla wartości atrybutu wierzchołkowego *Algorithm* zgodnie z domyślną wartością zdefiniowaną w tabeli 3.7. Przykładowe zastosowanie tego operatora zostało przedstawione na rysunku 3.13.

3.2.5.5 Operator mutacji poddrzewa

Operator mutacji poddrzewa, to operator, który ma zastosowanie do losowo wybranego wierzchołka n z wyłączeniem wierzchołków etykietowanych etykietą C, ENS_C lub FVC. Wybrany wierzchołek jest korzeniem poddrzewa, które będzie zastąpione nowym poddrzewem. Przebieg zastosowania tego operatora po wybraniu wierzchołka n jest następujący:

1. Przetransferuj niepuste atrybuty wierzchołkowe do rodzica wierzchołka n zaczynając od liści poddrzewa kończąc na wierzchołku n w ten sposób, że wartości poszczególnych atrybutów wierzchołkowych rodzica wierzchołka n są ich sumą z transferowanymi atrybutami wierzchołkowymi poszczególnych wierzchołków poddrzewa.

2. Usuń:.

   (a) Wartości z repozytorium hiperparametrów odpowiadające wierzchołkom w usuwanym poddrzewie.

   (b) Poddrzewo łącznie z jego korzeniem.

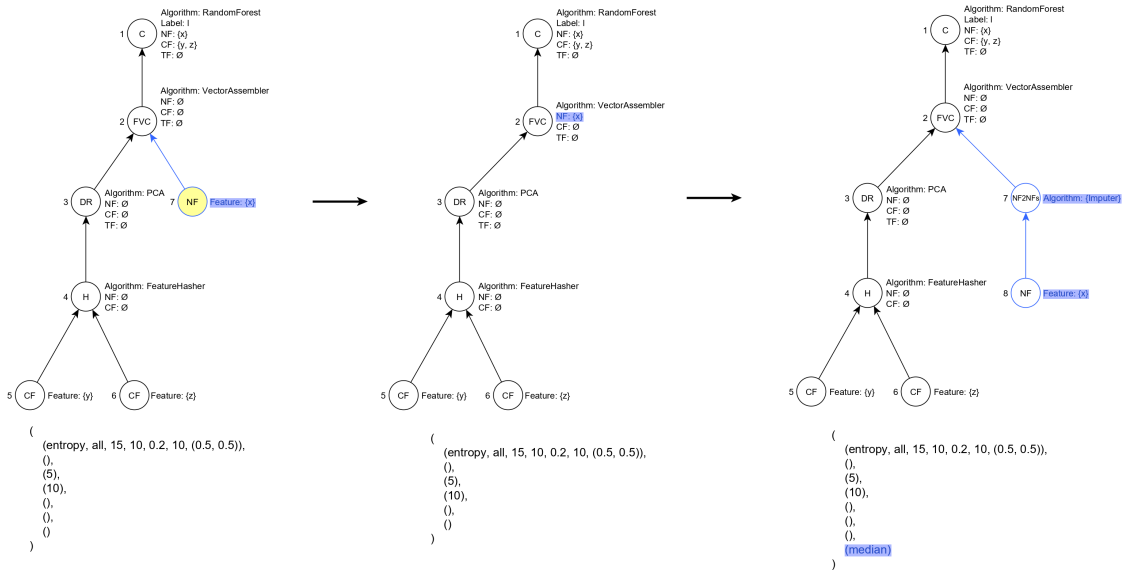
3. Przy pomocy zdefiniowanych produkcji gramatyki (def. 3.1.2.3) i diagramu sterującego (def. 3.1.3) stwórz nowe poddrzewo:

   (a) Wierzchołkiem początkowym diagramu sterującego jest losowy wierzchołek etykietowany produkcją, która w swojej lewej stronie zawiera wierzchołek etykietowany etykietą rodzica wierzchołka n.

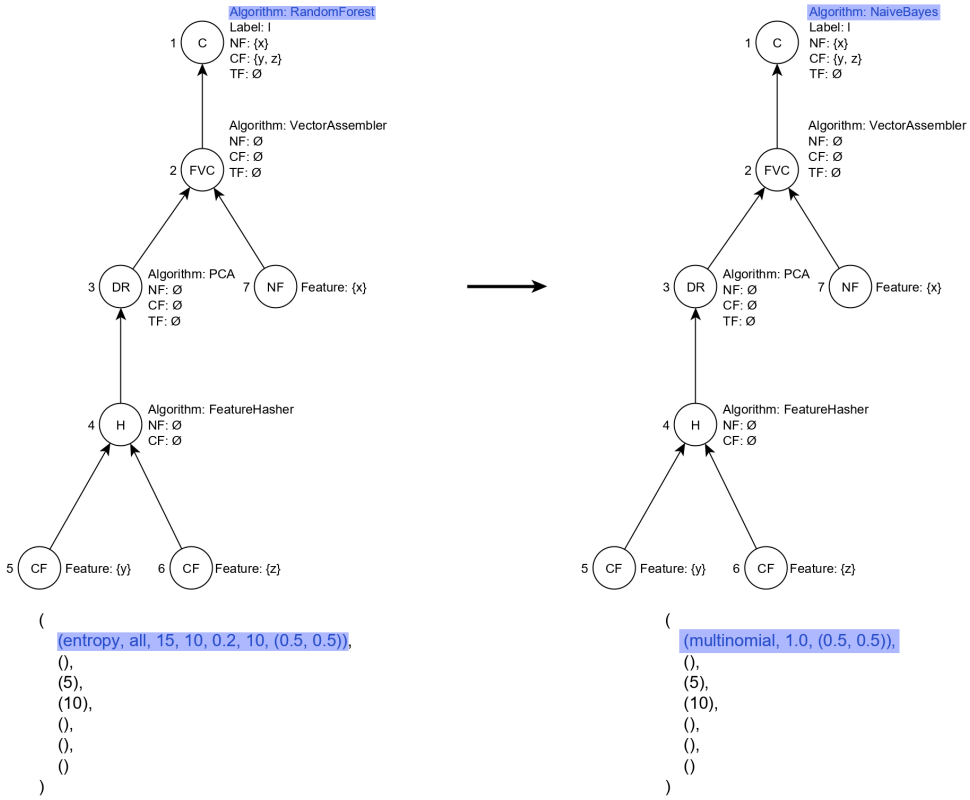
   (b) Stosuj produkcje do momentu osiągnięcia wierzchołka końcowego w diagramie sterującym.

   (c) Zgodnie z tabelą 3.7, dodaj do repozytorium hiperparametrów nowe, domyślne listy hiperparametrów odpowiadające nowym wierzchołkom w drzewie.

Przykładowe zastosowanie tego operatora zostało przedstawiona na rysunku 3.12.



Rysunek 3.12: Przykładowe zastosowanie operatora mutacji poddrzewa. Źródło: opracowanie własne



Rysunek 3.13: Przykładowe zastosowanie operatora mutacji wartości atrybutu wierzchołkowego *Algorithm* w danym wierzchołku i odpowiadającej mu listy hiperparametrów. Źródło: opracowanie własne

3.3 Algorytm jako całość

Algorytm budowy i optymalizacji modeli klasyfikujących w rozwiązaniu będącym przedmiotem pracy jest następujący:

1. Uruchom proces programowania genetycznego (def. 2.2.7) w oparciu o przygotowaną konfigurację, na którą składają się:
   - wybrana miara jakości będąca funkcją przystosowania osobników populacji (def. 2.4, 2.5),
   - komponenty (def. 2.11),
   - liczebność populacji,
   - liczba generacji,
   - prawdopodobieństwo krzyżowania,
   - prawdopodobieństwo mutacji,
   - operatory genetyczne (def. 3.2.5.1, 3.2.5.2, 3.2.5.3, 3.2.5.4, 3.2.5.5),
   - metoda selekcji osobników,
   - elitaryzm,
   - zbiór danych treningowych,
   - zbiór danych testowych.

   (a) W celu stworzenia populacji początkowej w procesie programowania genetycznego wykorzystaj zdefiniowaną gramatykę (def. 3.1), która umożliwia generowanie poprawnych drzew reprezentujących modele, które składają się na genotyp osobnika.

      i. Osobniki w populacji początkowej są stworzone dla konkretnego zestawu danych. Każde drzewo posiada przynajmniej $\frac{2}{3}$ wszystkich cech, które są zawarte w zbiorze danych uczących.

   (b) W oparciu o wygenerowane drzewa stwórz dla każdego z osobna repozytorium hiperparametrów (def. 3.2.2), które również składa się na genotyp osobnika.

   (c) Na podstawie genotypu osobnika stwórz program, który będzie zdolny do uczenia się przy pomocy zbioru danych treningowych. W wyniku uczenia powstanie model klasyfikujący, który poddany zostanie ewaluacji w ramach określonej miary jakości, będącej funkcją przystosowania, na zbiorze danych testowych.

Bibliografia

[Bun82] Horst Bunke. Attributed programmed graph grammars and their application to schematic diagram interpretation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(6):574–582, Nov 1982.

[Bun83] Horst Bunke. Graph grammars as a generative tool in image understanding. Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, redaktorzy, *Graph-Grammars and Their Application to Computer Science*, strony 8–19, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[Cra85] Nichael L. Cramer. A representation for the adaptive generation of simple sequential programs. *Proceedings of the 1st International Conference on Genetic Algorithms*, strony 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.

[DB15] Kiran Dahiya, Surbhi Bhatia. Customer churn analysis in telecom industry. *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, strony 1–6, Sep. 2015.

[dSPOP17] Alex Guimarães Cardoso de Sá, Walter José G. S. Pinto, Luiz Otávio Vilas Boas Oliveira, Gisele L. Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. *EuroGP*, wolumen 10196 serii *Lecture Notes in Computer Science*, strony 246–261, 2017.

[FKE+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, Frank Hutter. Efficient and robust automated machine learning. C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett, redaktorzy, *Advances in Neural Information Processing Systems 28*, strony 2962–2970. Curran Associates, Inc., 2015.

[Fla11] Mariusz Flasiński. *Wstęp do sztucznej inteligencji*. Wydawnictwo Naukowe PWN SA, Warszawa, 2011.

[GHLL06] Al Globus, Greg Hornby, Derek Linden, Jason Lohn. Automated antenna design with evolutionary algorithms, wrzesień 2006.

[Kot07] Sotiris B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in*

Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies, strony 3–24, Amsterdam, Holandia, 2007. IOS Press.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[KTH+17] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in weka. *J. Mach. Learn. Res.*, 18(1):826–830, Stycze/n 2017.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, wydanie 1, 1997.

[OBUM16] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, strony 485–492, New York, NY, USA, 2016. ACM.

[Oso13] Stanisław Osowski. *Metody i narzędzia eksploracji danych*. Wydawnictwo BTC, Legionowo, 2013.

[PLM08] Riccardo Poli, William B. Langdon, Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[Pol01] Riccardo Poli. General schema theory for genetic programming with subtree-swapping crossover. wolumen 2038, strony 143–159, 04 2001.

[Rut09] Leszek Rutkowski. *Metody i techniki sztucznej inteligencji*. Informatyka - Zastosowania. Wydawnictwo Naukowe PWN, 2009.

[Sam59a] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, 1959.

[Sam59b] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, Lipiec 1959.

[SL09] Marina Sokolova, Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, Lipiec 2009.

[SLL14] Ling Shao, Li Liu, Xuelong Li. Feature learning for image classification via multiobjective genetic programming. *Neural Networks and Learning Systems, IEEE Transactions on*, 25:1359–1371, lipiec 2014.

[Whi95] Peter A. Whigham. Grammatically-based genetic programming. Justinian P. Rosca, redaktor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, strony 33–41, Tahoe City, California, USA, 9 July 1995.

[Zah14] Matei Zaharia. *An Architecture for Fast and General Data Processing on Large Clusters*. Praca doktorska, EECS Department, University of California, Berkeley, Feb 2014.