Spis treści

1 Źródła

Na podstawie:

- Artificial Intelligence - Foundations of computational agents (książka w internecie)
- Fajny link do sieci neuronowych: `https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/`

2 Kartkówka 1 - Uczenie nadzorowane

2.1. Wiadomości podstawowe

2.1.1. ML workflow

Czyli ogólny cykl MachineLearning, przechodzimy przez kolejne fazy w celu uzyskania rezultatów.

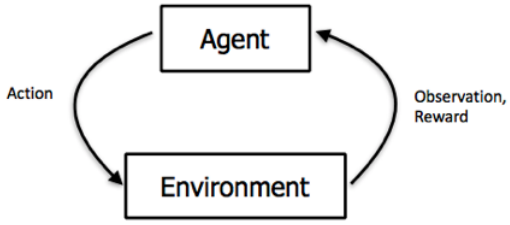
- Wstępne przygotowanie danych - zebranie danych, analiza danych, czyszczenie danych, normalizacja itd.
- Przygotowanie danych - integracja danych, filtrowanie, redukcja itd.
- Tworzenie zbiorów testowych
- Algorytmy i modele uczenia maszynowego - wybór, testowanie, dobieranie parametrów, wybór najlepszego
- Testy A/B i wdrożenie produkcyjne

2.1.2. Typy uczenia

Wyróżnia się wiele kategoryzacji uczenia, jednym z podziałów jest podział ze względu na sposób działania algorytmów:

- Uczenie nadzorowane (supervised learning) - mamy dane atrybuty wejściowe oraz atrybuty wyjściowe, a także zbiór trenujący, który zawiera konkretne dane z atrybutami wejściowymi i wyjściowymi. Zadanie polega na przewidzeniu wartości cech atrybutów wyjściowych mając dane wartości cech atrybutów wejściowych dla zbioru testowego. Formalnie problemy te nazywane są **klasyfikacja** (kiedy wartości atrybutów wyjściowych są dyskretne) oraz **regresja** (kiedy wartości atrybutów wyjściowych są ciągłe)
- Uczenie nienadzorowane (unsupervised learning) - w tych problemach nie mamy określonych atrybutów wyjściowych jak w uczeniu nadzorowanym. Inteligentny agent sam musi odkryć kategorie i wzory w analizowanych danych. Przykładową techniką stosowaną przez uczenie nienadzorowane jest klasteryzacja (klastry powstają na podstawie zauważenia podobieństwa pomiędzy atrybutami wejściowymi, nie ma jasno powiedziane dla jakich wartości wejściowych ma być dana wartość wyjściowa jak jest to w uczeniu nadzorowanym)
- Uczenie przez wzmacnianie (reinforcement learning) - w tych problemach inteligentny agent próbuje wykonywać jakieś akcje w celu uzyskania danych, a następnie określa czy podjęta akcja była korzystna czy nie (punishment lub reward). Podejście takie jest w niektórych przypadkach konieczne, wyobraźmy sobie np., że chcemy nauczyć robota chwytać przedmiot - bardzo trudno będzie nam

stworzyć zbiór danych, który będzie zawierał wszystkie możliwe współrzędne oraz wszystkie możliwe ruchy. W takim przypadkach chcemy aby inteligentny agent sam potrafił zebrać dane i ocenić poprawność swoich akcji.



- Inne - istnieje wiele innych, które jak na razie z naszego punktu widzenia są mniej ważne

Kolejnym podziałem jest podział uwzględniający dostęp do danych:

- offline - wszystkie dane ze zbioru trenującego są dostępne dla agenta zanim ten ma podjąć jakąś decyzje
- online - dane przychodzą podczas gdy agent się uczy
- active - to rodzaj uczenia online, które polega na tym, że agent sam stwierdza, które przykłady byłyby dla niego użyteczne i stara się je uzyskać

2.1.3. Zbiór uczący(trenujący, training), testowy i walidacyjny

- uczący - zbiór danych na którym uczy się agent, zawiera zarówno wartości atrybutów wejściowych jak i wyjściowych
- zbiór testowy - zbiór danych na którym testowane jest czego agent się nauczył, zawiera tylko wartości dla atrybutów wejściowych
- zbiór walidacyjny - używany jest w celu zbadania czy zbudowany model działa poprawnie i dostosowania jego parametrów

Zbiory te powinny być niezależne, zbiór testowy ma służyć do sprawdzenia jak dobrze inteligentny agent potrafi podejmować decyzje.

2.1.4. Reprezentacja

Każdy inteligentny agent musi posiadać jakąś reprezentacje danych na podstawie których ma podejmować decyzje. Im więcej danych zawiera reprezentacja, tym bardziej użyteczna do tworzenia rozwiązań.

Z drugiej strony im większa reprezentacja tym trudniej się uczyć, ponieważ wymaga to operacji na dużej liczbie danych. Konieczne jest więc znalezienie kompromisu.

Machine learning jest najczęściej omawiany w kontekście konkretnej reprezentacji, ważne jest więc aby znać podstawowe z nich.

2.1.5. Noise

Noise to termin odnoszący się do pewnych zaburzeń w danych (np. brak części danych, część danych nieprawidłowa itp.), jedną z pierwszych faz w machine learningu jest czyszczenie danych, które to polega między innymi na usuwaniu hałasu z danych, można to zrobić między innymi przy pomocy interpolacji (przewidywanie wartości która znajduje się pomiędzy innymi) lub ekstrapolacji (przewidywanie wartości, które jeszcze nie są znane)

2.2. Uczenie nadzorowane

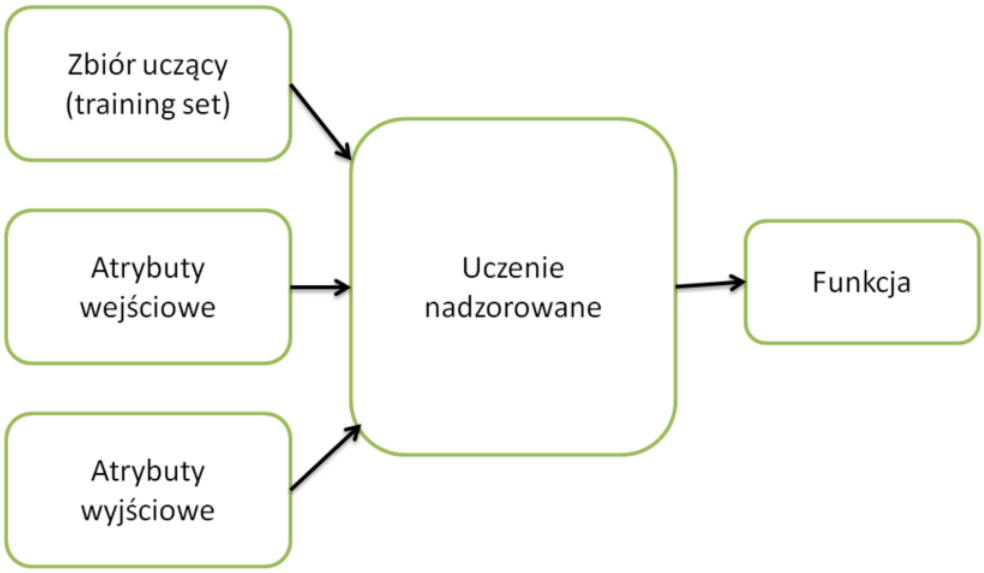
Na wejściu procesu uczenia nadzorowanego jest:

- zbiór atrybutów, a wśród nich podział na
  - wejściowe
  - wyjściowe
- dziedziny atrybutów - każdy z atrybutów ma swoją dziedzinę, czyli zbiór wartości jakie może przyjmować, dziedzina może być dyskretna (zbiór skończony) lub ciągła
- zbiór przykładów, czyli zebrane dane, które dzielimy na
  - zbiór uczący - w którym wartości są znane zarówno dla atrybutów wejściowych jak i wyjściowych
  - zbiór testujący - w którym wartości są znane tylko dla atrybutów wejściowych

Celem uczenia nadzorowanego jest przewidzieć wartości atrybutów wyjściowych zbioru testowego na podstawie atrybutów wejściowych.

2.3. Modele uczenia nadzorowanego

Nauczony model to po prostu funkcja, która bierze na wejściu określone wartości atrybutów wejściowych i zwraca na wyjściu pewne wartości atrybutów wyjściowych.



Poniżej omówione zostaną podstawowe modele dla uczenia nadzorowanego.

2.3.1. Drzewa decyzyjne (decision trees)

Służy do realizacji uczenia nadzorowanego, które oparte jest na klasyfikacji (dyskretne dziedziny atrybutów). Jest to jedna z podstawowych i najprostszych technik. Drzewo takie tak naprawdę reprezentuje funkcję dyskretną, która przyporządkowuje dyskretnym wartościom atrybutów wejściowych dyskretne wartości atrybutów wyjściowych.
Drzewo decyzyjne to drzewo, które:

- każdy węzeł, który nie jest liściem ma etykietę będącą warunkiem
- każdy węzeł, który nie jest liściem ma dwoje dzieci, jeden z etykietą *true*, a drugi z etykietą *false*
- każdy liść zawiera wartość atrybutu wyjściowego

Przykładowe dane i drzewo decyzyjne utworzone na ich podstawie:

Example	*Author*	*Thread*	*Length*	*Where_read*	*User_action*
e_1	*known*	*new*	*long*	*home*	*skips*
e_2	*unknown*	*new*	*short*	*work*	*reads*
e_3	*unknown*	*followup*	*long*	*work*	*skips*
e_4	*known*	*followup*	*long*	*home*	*skips*
e_5	*known*	*new*	*short*	*home*	*reads*
e_6	*known*	*followup*	*long*	*work*	*skips*

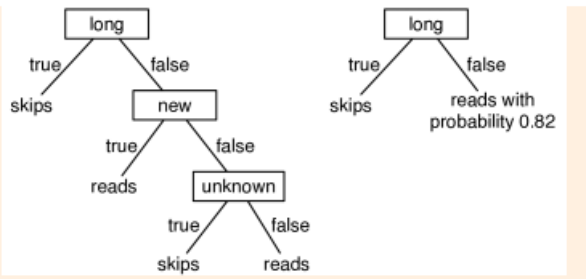


Figure 7.6: Two decision trees

Drzewo na prawo podejmuje decyzje na podstawie prawdopodobieństwa (zna tylko jedną wartość dla atrybutu long), a drzewa na lewo pozwala w sposób deterministyczny określić wartość atrybutu wyjściowego, bierze pod uwagę wszystkie atrybuty.

2.3.2. Regresja liniowa (linear regression)

Regresja liniowa to model, który pozwala dopasować funkcję liniową do zbioru danych uczących w przypadku gdy dziedzina atrybutów jest ciągła.

Suppose the input features, $X_1, \ldots, X_n$, are all numeric and there is a single target feature Y. A **linear function** of the input features is a function of the form

$$\begin{aligned} \widehat{Y}^{\overline{w}}(e) &= w_0 + w_1 * X_1(e) + \ldots + w_n * X_n(e) \\ &= \sum_{i=0}^{n} w_i * X_i(e) \end{aligned}$$

Problem, który teraz powstaje jest oczywiście jak znaleźć wektor wag w tym równaniu, z pomocą przychodzi nam tutaj równanie, które opisuje jak dobrze funkcja wpasowuje się w dane, czyli równanie opisujące sumę kwadratów błędu, minimalizacja takiej funkcji pozwala na znalezienie wektora wag.

$$\begin{aligned} error(E, \overline{w}) &= \sum_{e \in E} \left(Y(e) - \widehat{Y}^{\overline{w}}(e)\right)^2 \\ &= \sum_{e \in E} \left(Y(e) - \sum_{i=0}^{n} w_i * X_i(e)\right)^2. \end{aligned} \tag{7.1}$$

Nie w każdym przypadku możliwe jest jednak obliczenie wartości tych wag poprzez minimalizacje błędu, czasami konieczne jest zastosowanie innych metod iteracyjnych, jedną z takich metod jest metoda gradientowa.

Metoda gradientowa to metoda iteracyjna pozwalająca na znalezienie minimum funkcji. Metoda ta bazuje na obliczaniu pochodnych cząstkowych po każdej z wag i zmianie wartości wag proporcjonalnie do obliczonych wartości pochodnych.

2.4. Nadmierne dopasowanie, przeuczenie, przetrenowanie (overfitting)

Nadmierne dopasowanie to sytuacja w której agent podejmuje poprawne decyzje na zbiorze uczącym, ale nie podejmuje poprawnych decyzji na zbiorze testowym lub innym z którego weźmiemy dane. Oznacza to, że agent został nauczony tylko na tych konkretnych danych, które znajdywały się w zbiorze uczącym i na nich działa dobrze, ale nie przekłada się to na ogólnie poprawne funkcjonowanie algorytmu dla ogólnych danych. Można więc powiedzieć, że model 'działa' tylko w pewnym szczególnym przypadku, a tak naprawdę nie działa wcale, ponieważ nie robi tego co powinien na danych ogólnych.

Powodami nadmiernego dopasowania mogą być:

- niepoprawnie przygotowanie danych wejściowych - obecne w danych noise, randomness
- zbyt mała liczba przykładów w zbiorze wejściowym - np. jedna ocena 5-gwiazdkowa, to za mało aby orzec o 5-gwiazdkowej jakości
- zbyt duża liczba przykładów w zbiorze wejściowym - gdy każdy przykład ma inne wartości, trudno jest znaleźć jakąkolwiek zależność pomiędzy nimi, np. mamy mnóstwo danych o pacjentach, ale każdy z nich ma inne dolegliwości

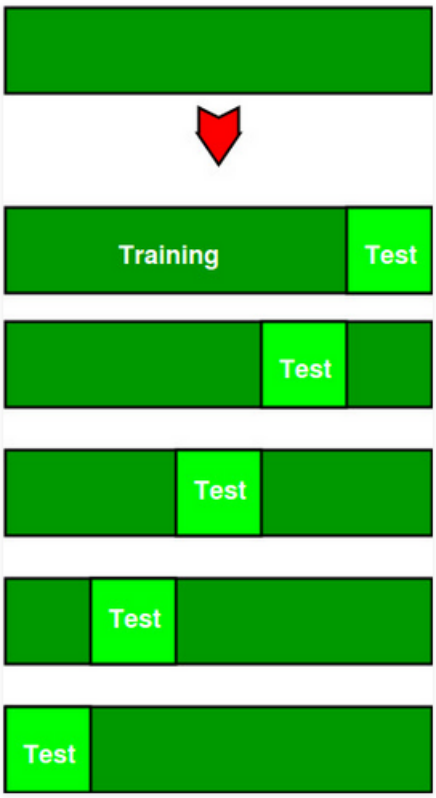
- złożoność modelu - przykładowo, stosując regresje, nie zawsze wielomian o wyższym stopniu będzie lepszy

2.4.1. Metody zapobiegania nadmiernemu dopasowaniu

Istnieje wiele metod których celem jest zapobieganie nadmiernemu dopasowaniu. Jedną z najbardziej popularnych jest walidacja krzyżowa.

Walidacja krzyżowa polega na podziale zbioru uczącego na dwie równe części i użycie jednej z części jako zbioru testowego. Wadą tej metody jest jednak to, że taki podział powoduje znaczną utratę danych na których uczy się model (aż połowa !) dlatego częściej stosowanym rozwiązaniem jest metoda k-złożeń walidacji krzyżowych (k-fold cross validation).

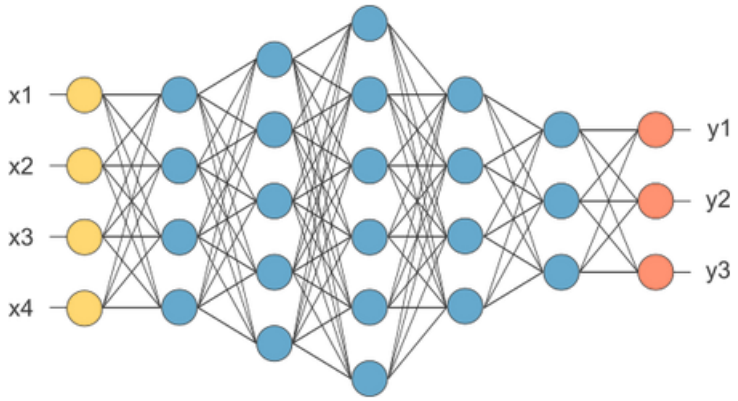
W metodzie k-cross validation zbiór uczący znów dzielimy na części, ale tym razem jest ich *k*. Następnie przeprowadzamy uczenie na *k-1* z nich, a pozostały jeden stosujemy do walidacji. Czynność tą powtarzamy dla wszystkich możliwych kombinacji.



2.5. Sieci neuronowe

Sieć neuronowa to model, który został zainspirowany biologiczną budową mózgu. Ma architekturę sieci składającej się z warstw, warstwy z kolei składają się z neuronów, graficznie przedstawiamy neurony jako koła. Cała struktura zawiera parametry, które są dostosowywane w celu wykonania pewnego zadania.
Pierwszą warstwą jest zawsze warstwa wejść, neurony w tej warstwie zawierają zawsze tylko jedną liczbę

rzeczywistą o wartości pomiędzy 0 i 1.
Ostatnią warstwą jest zawsze warstwa wyjściowa, tutaj neurony również zawierają zawsze tylko jedną liczbę pomiędzy 0 i 1.
Wszystkie warstwy pośrednie nazywane są warstwami ukrytymi.



Połączenia pomiędzy warstwami mają przypisane wagi, które są dowolnymi liczbami rzeczywistymi, wartość z neuronu z poprzedzającej warstwy jest mnożona przez wagę, operacja ta jest wykonywana dla wszystkich neuronów, które mają połączenia z neuronem w następnej warstwie, wartości wynikowe są dodawane do siebie i tworzą sumę.

Oczywiście suma ta może być dowolną wartością, nie musi być z przedziału 0 do 1, dlatego stosuje się specjalną funkcję, która skaluje sumę tak aby mogła być przechowywana w neuronie w następnej warstwie.

Ostatecznie dostajemy liczbę rzeczywistą pomiędzy 0 i 1 w neuronie wynikowym i ta liczba określa prawdopodobieństwo, że coś można zakwalifikować do danej klasy (takiej którą reprezentuje dany neuron).

Cała sztuka polega na odpowiednim dobraniu wszystkich wag, tak aby przy odpowiednich danych wejściowych, otrzymywać poprawną klasyfikację. Proces dobierania wag dla sieci neuronowej nazywamy trenowaniem/uczeniem sieci. Proces ten realizowany jest na podstawie poprawnych klasyfikacji, które są już nam znane (w uczeniu nadzorowanym), ale sieci neuronowe mogą być także stosowane w innych rodzajach uczenia.

2.6. Backpropagation

To algorytm, który jest stosowany do efektywnego uczenia się sieci neuronowych, opiera się o zastosowanie metody gradientowej.

3 Kartkówka 2 - Uczenie nienadzorowane

3.1. Uczenie nienadzorowane

W tym rodzaju uczenia nie mamy danych atrybutów wyjściowych dla zbioru uczącego, a jedynie dane w tym zbiorze. Zadaniem uczenia nienadzorowanego jest odkrycie klasyfikacji jakiej można dokonać na danych.

3.2. Klasteryzacja

Jedną ogólną metodą do realizacji uczenia nienadzorowanego jest klasteryzacja, metoda ta dzieli przykłady ze zbioru danych na klastry (klasy). Klasa jest przewidywaniem wartości dla przykładów z tej klasy. Każda klasteryzacja obarczona jest błędem predykcji, najlepsza klasteryzacja to taka, która minimalizuje ten błąd. Klasteryzacja jest tylko ogólnym schematem postępowania, konkretne metody implementują ten schemat postępowania (np. k-Means).

3.2.1. Hard clustering

W *hard clustering* każdy przykład jest definitywnie umieszczany w pewnej klasie. Ta klasa jest następnie używana do przewidzenia przyszłych wartości tego przykładu.

3.2.2. Soft clustering

W *soft clustering* każdy z przykładów ma rozkład prawdopodobieństwa dla przynależności do danej klasy. Przewidywanie przyszłych wartości dla danego przykładu jest średnią ważoną przewidywań dla klas w których znajduje się dany przykład, wagami tej średniej są prawdopodobieństwa, że przykład znajduje się w danej klasie.

3.3. k-Means

- realizuje założenia silnej klasteryzacji (hard clustering)
- wejście do algorytmu są przykłady ze zbioru uczącego oraz liczba klas k
- wyjściem jest funkcja, która każdemu przykładowi przyporządkowuje klasę
- każda z klas ma przypisane do niej wartości atrybutów
- minimalizacja błędu polega na znalezieniu klas oraz przypisania ich do przykładów w takim sposób aby różnice pomiędzy wartościami dla klas i przykładów były jak najmniejsze

3.3.1. Działania k-Means

- Na początku algorytm randomowo przydziela klasy do przykładów
- Następnie przechodzi w iteracyjną procedurę, której celem jest minimalizacja błędu:
  - algorytm oblicza 'odległości' (błąd) przykładów od danej klasy
  - na tej podstawie wartości atrybutów w klasie są aktualizowane
  - itd. aż do osiągnięcia braku poprawy

3.4. EM - Expectation Maximisation

- realizuje założenia słabej klasteryzacji (soft clustering)
- algorytm nie przypisuje definitywnie przykładów do grup jak robił to k-Means, ale estymuje prawdopodobieństwo, że dany przykład należy do danej klasy
- algorytm korzysta z założeń o wielowymiarowym rozkładzie normalnym danych

3.4.1. Działanie EM

- Dla aktualnych parametrów rozkładu danych, przypisz prawdopodobieństwo przynależności do grup
- Zamień aktualne parametry rozkładu na bliższe zgodności z danymi korzystając z prawdopodobieństwa przynależności z poprzedniego kroku

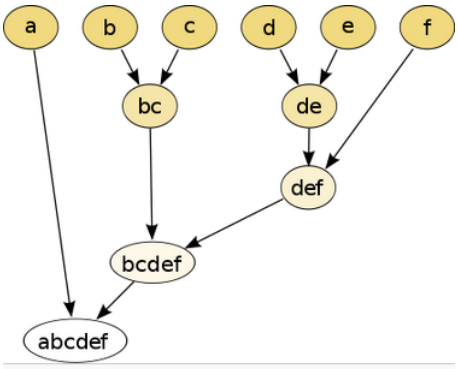
3.4.2. Mieszanki gaussowskie

To rozkład prawdopodobieństwa stosowany dla ciągłych zbiorów danych, stosowany jest w metodzie EM.

3.5. Klasteryzacja hierarchiczna

- Algorytm klasteryzacji, który nie zakłada z góry liczby klastrów
- Początkowo każdy przykład jest osobnym klastrem i są one łączone aż do uzyskania jednego klastra (procedura aglomeracyjna) lub w drugim podejściu:
- Początkowo wszystkie przykłady tworzą jeden klaster, który następnie jest dzielony aż do uzyskania oddzielnego klastra z każdego przykładu
- Dzielenie i łączenie odbywa się na podstawie tzw. macierzy odległości (podobieństw), która zwiera odległości pomiędzy klastrami
- Najbliżej położone względem siebie klastry są razem łączone

Proces klasteryzacji hierarchicznej przedstawia się często w postaci tzw. dendogramu:



Przedstawienie podziału na klastry (procedura aglomeracyjna)

3.6. Reguły asocjacyjne (analiza koszykowa) i algorytmy ich generacji

PODSTAWOWE OZNACZENIA

- Pozycje (ang. items) opisują dostępne towary. Zakłada się, że zbiorem wszystkich towarów jest $I = \{i_1, i_2, ..., i_m\} \quad (items)$
- baza transakcji $D = \{(tid_1, T_1), (tid_2, T_2)...\}$ zawiera transakcje jako pary (tid_j, T_j), gdzie:
  - tid_j : unikalny identyfikator
  - $T_j \subset I$: zbiór zakupionych towarów .
- itemset: każdy podzbiór zbioru towarów I;
- k-itemset: podzbiór o k elementach.
- $s(X)$ - liczba transakcji zawierających itemset X.

DEFINITION

Regułą asocjacyjną nazywamy każdą implikację typu

$$X \implies Y$$

gdzie X, Y są itemsetami. Jakość takiej reguły mierzymy jest funkcjami:

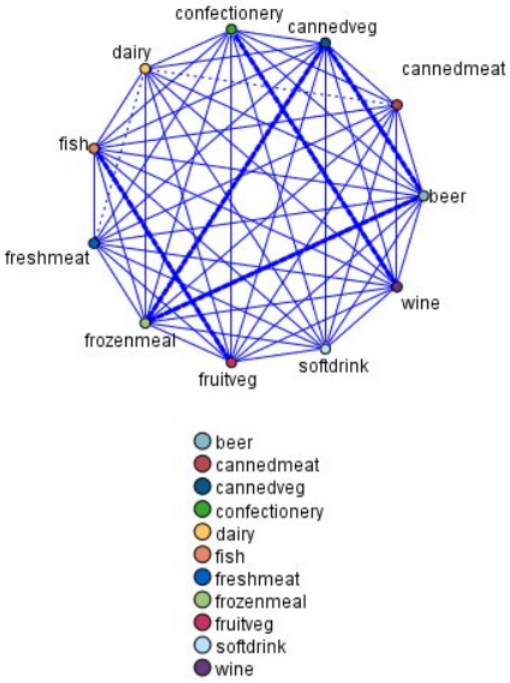
- wsparcie (support)

$$support(X \implies Y) = s(X \cup Y)$$

- wiarygodność (confidence)

$$confidence(X \implies Y) = \frac{s(X \cup Y)}{s(X)}$$

Reguły asocjacyjne kojarzą konkretny wniosek (np. decyzję o zakupie konkretnego produktu) ze zbiorem warunków (np. zakupem kilku innych produktów). Na przykładzie z obrazka jedną z reguł może być to, że beer często występuje wtedy, gdy jednocześnie występują cannedveg i frozenmeal.



3.6.1. Wsparcie i wiarygodność

LISTA AUTORÓW (ITEMS)

A	Jane Austen
C	Agatha Christie
D	Sir Arthur Conan Doyle
T	Mark Twain
W	G. Wodehouse

TRANSAKCJE

TID	Kupione książki
10	A C T W
20	C D W
30	A C T W
40	A C D W
50	A C D T W
60	C D T

Reguły	wsparcie	st. wiar.
$A \Longrightarrow C$	4	100%
$C \Longrightarrow W$	5	83,3%
$AC \Longrightarrow T$	4	75%
$T \Longrightarrow ACW$	3	75%

PROBLEM

DANE SĄ:

- zbiór pozycji $I = \{i_1, i_2, ..., i_m\}$
- baza transakcji $D = \{(tid_1, T_1), (tid_2, T_2)...\}$
- stałe sup_min = minimalna wartość wsparcia i $conf\_min$ = minimalny stopień wiarygodności

PROBLEM: Znaleźć wszystkie reguły asocjacyjne o
- wsparciu $\geq sup\_min$
- stopniu wiarygodności $\geq conf\_min$

3.6.2. Algorytmy znajdywania reguł asocjacyjnych

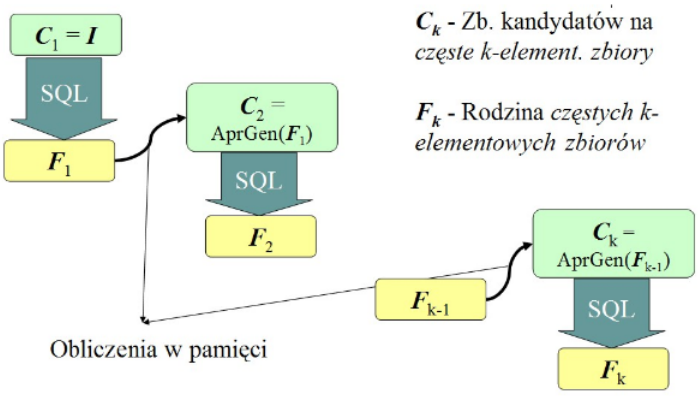
Algorytmy wykorzystywane do znajdowania reguł stosują metodę generowania i testowania — początkowo generują proste reguły i walidują je względem zbioru danych. Dobre reguły są zachowywane, a wszystkie reguły, z zachowaniem różnych ograniczeń, podlegają specjalizacji. Specjalizacja polega na dodawaniu warunków do reguły. Uzyskane nowe reguły są walidowane względem danych i proces znów zapisuje najlepsze i najbardziej interesujące reguły.

3.6.3. Algorytm apriori

Jeden z najwydajniejszych i najbardziej popularnych algorytmów generowania reguł asocjacyjnych (analizy koszykowej). Funkcje implementujące ten algorytm posiadają dwa główne kroki:

Łączenie: do zbioru C_k wstawiamy sumy takich par $X, Y \in F_{k-1}$, które mają wspólne k-2 początkowych elementów np.
dla $F_{k-1} = AB, AC, AD, AE, BC, BD, BE$
mamy: $C_k = ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE$

Obcinanie: Celem operacji jest redukowanie rozmiaru zbioru C_k przed sprawdzaniem ich wsparcia w bazie transakcji. W tym celu wykorzystujemy własciwość Apriori, z której wynika, że jeśli jakiś (k-1)-podzbiór danego kandydata nie występuje w F_{k-1}, to ten kandydat powinien być usunięty z C_k.
otrzymujemy zatem: $C_k = ABC, ABD, ABE$



3.7. Inne typy learningu

3.7.1. Transfer learning

Uczenie transferowe stosujemy, gdy chcemy wykorzystać już istniejący model w odniesieniu do nowego, powiązanego problemu. Przykład: algorytm rozpoznający przechodniów na nocnych zdjęciach wykorzystujemy do rozpoznawania ich także na zdjęciach w oświetleniu dziennym. Zadanie to byłoby utrudnione lub niemożliwe z wykorzystaniem tradycyjnych technik nadzorowanych uczenia maszynowego.

3.7.2. Ensemble learning

Technika uczenia maszynowego łącząca kilka algorytmów w celu osiągnięcia modelu o lepszej wydajności.

W przeciwieństwie do tradycyjnych modeli uczenia nadzorowanego, gdzie algorytm przeszukuje zestaw hipotez w celu znalezienia najbardziej optymalnej, ensemble learning pozwala na grupowanie kilku hipotez w celu utworzenia jednej, w założeniu lepszej od pozostałych.

Podstawą działania są drzewa decyzyjne określające definicje i wskazujące metody zespołowe. Drzewo takie określa wartość decyzją na podstawie serii pytań i warunków, a na następnie rozważa czynniki i podejmuje decyzje albo zadaje kolejne pytanie, dotyczące każdego z nich.

4 Kartkówka 3 - Reprezentacja niepewności

4.1. Opis problemu

Problem występuje w przypadkach gdy inteligentny agent nie jest pewien na 100%, że dane informacje są prawdziwe. Zamiast tego agent przyjmuje, że dane informacje są prawdziwe z pewnym prawdopodobieństwem.

Przykładem takiej sytuacji może być diagnozowanie pacjenta. Lekarz nigdy nie wie w 100% co się dzieje w środku pacjenta (no nie zagląnie tam i nie sprawdzi) dlatego pewne założenia (pewne informacje, pewną wiedzą) przyjmuje z jakimś prawdopodobieństwem.

We wszystkich tego rodzaju przypadkach mamy do czynienia z niepewnością. Niepewność może wynikać np. z niemożliwości uzyskania całkowitej informacji lub niedeterminizmu. Do formalnej reprezentacji niepewności stosujemy teorie prawdopodobieństwa.

4.2. Stan wiary (belief state)

Pojęcie to określa jeden ze sposobów radzenia sobie z niepewnością w podejmowaniu decyzji przez inteligentnego agenta. W tym podejściu należy zdefiniować wszystkie możliwe stany w jakich może znaleźć się agent, a następnie w każdym z tych stanów każdej tezie przypisujemy pewne prawdopodobieństwo. Metoda ta jest jednak mało praktyczna ze względu na konieczność reprezentacji wszystkich możliwych stanów agenta, co jest rzadko możliwe w praktyce.

4.3. Przypomnienie podstaw prawdopodobieństwa

4.3.1. Obserwacja ω

Obserwacja to po prostu pewien zaobserwowany fakt z rzeczywistości, którą opisujemy, np. "W rzucie kostką wypadło 6 oczek". Pojedynczą obserwację oznaczamy symbolem ω. Rozpatrując dane zagadnienie mamy wiele obserwacji, przykładowo w rzucie kostką może wypaść 1,2,3,4,5 lub 6 oczek. Każdy z tych przypadków jest oddzielną obserwacją, a zbiór ich wszystkich oznaczamy symbolem Ω.

4.3.2. Miara prawdopodobieństwa $P(\omega)$

Miara prawdopodobieństwa to funkcja P mapująca obserwacje na dowolne nieujemne liczby rzeczywiste (chociaż najczęściej przyjętą konwencją jest przedział [0;1]), taka że:

$$\sum_{w \in \Omega} P(w) = 1$$

4.3.3. Zmienna losowa *X*

Jeśli mamy jakieś zjawisko to często chcielibyśmy przyjąć pewną reprezentację zjawisk, które w nim zachodzą. Przykładowo w rzucie kostką, wynik każdego z rzutów chcielibyśmy zapisać za pomocą jednej liczby 1,2,3,4,5 lub 6, tak aby nie musieć określać za każdym razem wyniku jako 'Wypadło 3 oczka'. Do tego właśnie służy zmienna losowa, każdej z obserwacji przypisujemy jakąś wartość z dziedziny tej zmiennej. Zmienne losowe oznaczmy dużymi literami, przykładowo *X, Y*. Dziedzina zmiennej losowej może być dyskretna lub ciągła.

4.3.4. Rozkład prawdopodobieństwa zmiennej losowej *P(X)*

To funkcja, która określa prawdopodobieństwo, że zmienna losowa przyjmie określoną wartość ze swojej dziedziny. Np. $P(X = 1) = \frac{1}{6}$, oznacza, że prawdopodobieństwo, że zmienna losowa będzie miała wartość 1 wynosi $\frac{1}{6}$.

4.3.5. Prawdopodobieństwo warunkowe

Prawdopodobieństwo warunkowe $P(h|e)$ to prawdopodobieństwo prawdziwości formuły h przy znanym fakcie (wartości) e. Wyprowadzenie wzoru na prawdopodobieństwo warunkowe:

$$
\begin{aligned}
1 &= \sum_{w} P(w \mid e) \\
&= \sum_{w \,:\, e \text{ is true in } w} P(w \mid e) + \sum_{w \,:\, e \text{ is false in } w} P(w \mid e) \\
&= \sum_{w \,:\, e \text{ is true in } w} c * P(w) + 0 \\
&= c * P(e)
\end{aligned}
$$

4.3.6. Reguła Bayesa

Reguła Bayes'a [edytuj]

Agent, który dowie się o nowym fakcie, musi zaktualizować obliczone prawdopodobieństwa. Reguła Bayes'a opisuje, jak to zrobić. Załóżmy, że agent obliczył P(*h* | *k*), a następnie zaobserwował fakt ***e***. Wtedy reguła Bayes'a wygląda następująco (pod warunkiem, że P(***e*** | *k*) ≠ 0):

$$P(h \mid e \wedge k) = \frac{P(e \mid h \wedge k) * P(h \mid k)}{P(e \mid k)}.$$

Co często jest zapisywane z pominięciem *k* (zakładając, że P(*h*) to obecny stan wiedzy agenta o prawdopodobieństwie *h* z uwzględnieniem wszystkich znanych faktów):

$$P(h \mid e) = \frac{P(e \mid h) * P(h)}{P(e)}.$$

4.3.7. Wartość oczekiwana

Wartość oczekiwana [edytuj]

Wartość oczekiwana jakieś funkcji to suma wartości tej funkcji dla każdej z obserwacji, pomnożonych przez prawdopodobieństwo tych obserwacji:

$$\mathscr{E}_P(f) = \sum_{\omega \in \Omega} f(\omega) * P(\omega)$$

W wypadku, gdy α jest formułą, a f jest funkcją przyjmującą 1 gdy α jest prawdziwe, a 0, jeśli nie, to $\varepsilon_P(f) = P(\alpha)$. Przykład działania: przykład 8.9 ze strony https://artint.info/2e/html/ArtInt2e.Ch8.S1.SS4.html

4.4. Niezależność

Używając tylko standardowych aksjomatów o prawdopodobieństwie, pełny rozkład prawdopodobieństwa dla n zmiennych, wymagany do obliczania prawdopodobieństw warunkowych, składa się z 2^{n-1} wartości.

Aby zmniejszyć tę liczbę należy wykorzystać niezależność warunkową, czyli założenie, że każda zmienna zależy bezpośrednio tylko od kilku innych.

Użycie twierdzeń o niezależności może radykalnie zmniejszyć ilość informacji niezbędnych do określenia pełnego rozkładu. Jeśli pełny zbiór zmiennych można podzielić na niezależne podzbiory, wtedy pełny rozkład można rozłożyć na osobne rozkłady tych podzbiorów.

4.5. Niezależność warunkowa

Niezależność warunkowa [edytuj]

Zmienna losowa X jest warunkowo niezależna od zmiennej losowej Y przy danym zbiorze zmiennych losowych Zs, gdy:

$P(X \mid Y,Zs) = P(X \mid Zs)$

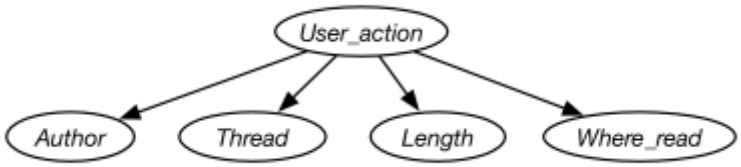
4.6. Niezależność bezwarunkowa

Niezależność bezwarunkowa [edytuj]

Zmienne losowe X i Y są niezależne bezwarunkowo gdy P(X,Y)=P(X)*P(Y). Jest to ekwiwalentem tego, że są warunkowo niezależne przy braku danych założeń, co nie musi znaczyć, że będą warunkowo niezależne przy danych jakichś założeniach.

4.7. Naiwny klasyfikator Bayesa NBC

Naiwny klasyfikator Bayesa to klasyfikator statystyczny, który działa przy założeniu, że parametry wejściowe są warunkowo niezależne od siebie nawzajem pod warunkiem parametru wyjściowego. Można to przedstawić za pomocą sieci Bayesowskiej, w której parametr wyjściowy nie ma żadnych rodziców, a każdy parametr wejściowy ma tylko jednego rodzica - parametr wyjściowy. Model ten wymaga znajomości prawdopodobieństwa $P(Y)$ parametru wyjściowego Y i prawdopodobieństw warunkowych $P(X_i|Y)$ każdego z parametrów wejściowych X_i.



Powyższy model może służyć do przewidzenia akcji użytkownika (czy przeczyta artykuł) przy znanych: autorze, wątku/temacie, długości i miejscu czytania jakiegoś artykułu.

4.8. Sieci Bayesowskie

Sieci bayesowskie są rodzajem probabilistycznego modelu graficznego, który wykorzystuje wnioskowanie bayesowskie do obliczeń prawdopodobieństwa. Sieci bayesowskie mają na celu modelowanie zależności warunkowej poprzez reprezentowanie jej przez krawędzie na grafie kierunkowym.

Bardziej formalna definicja:

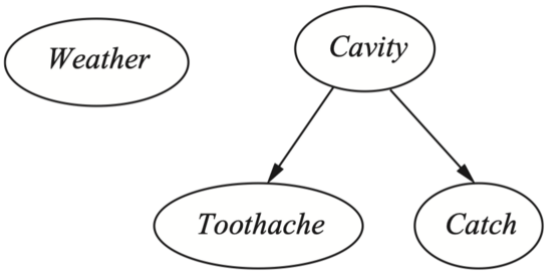
Sieć bayesowska to skierowany graf acykliczny, w którym każda krawędź odpowiada warunkowej zależności, a każdy węzeł odpowiada unikalnej zmiennej losowej:

- Każdy węzeł odpowiada zmiennej losowej, która może być dyskretna lub ciągła.
- Jeśli istnieje krawędź od węzła X do węzła Y, mówi się, że X jest rodzicem Y. Graf ten jest skierowanym grafem acyklicznym, czyli takim który nie posiada cyklów skierowanych
- Każdy węzeł X_i ma warunkowy rozkład prawdopodobieństwa $P(X_i|Parents(X_i))$, który określa wpływ rodziców na dany węzeł.

Znaczenie strzałki z węzła X do Y jest zwykle takie, że X ma bezpośredni wpływ na Y, co sugeruje, że "przyczyny"powinny być rodzicami efektów. Po ustaleniu topologii sieci Bayesowskiej musimy jedynie określić warunkowy rozkład prawdopodobieństwa dla każdej zmiennej, biorąc pod uwagę jej rodziców.

4.8.1. Przykład 1

W tym przykładzie zmienna Weather jest niezależna od innych zmiennych a Toothache jest warunkowo niezależny od Catch.



Formalnie na warunkową niezależność Toothache i Catch od Cavity, wskazuje na brak powiązania między nimi (Toothache i Cavity). Z tego prostego przykładu możemy wywnioskować, że

- Cavity jest bezpośrednią przyczyną Tootchache i Catch
- Nie ma bezpośredniego związku przyczynowego między Toothache a Catch.

4.8.2. Przykład 2

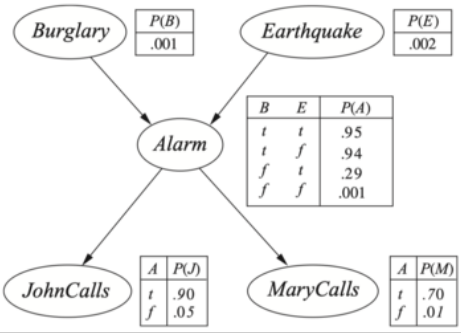


Figure 14.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

W domu zainstalowano nowy alarm antywłamaniowy. Jest dość niezawodny w wykrywaniu włamania, ale czasami reaguje również na niewielkie trzęsienia ziemi. Masz również dwóch sąsiadów, Johna i Mary, którzy obiecali zadzwonić do ciebie w pracy, gdy usłyszą alarm. John prawie zawsze dzwoni, gdy słyszy alarm, ale czasami myli dzwonek telefonu z budzikiem i wtedy też dzwoni. Mary natomiast lubi głośną muzykę i często zupełnie nie reaguje na alarm. Biorąc pod uwagę historię tego kto dzwonił lub nie dzwonił, chcielibyśmy oszacować prawdopodobieństwo włamania.

Struktura sieci na obrazku obok, pokazuje, że włamanie i trzęsienia ziemi bezpośrednio wpływają na prawdopodobieństwo uruchomienia alarmu, ale to, czy John i Mary zadzwonią, zależy tylko od alarmu. Sieć reprezentuje zatem nasze założenia, że Mary i John nie dostrzegają bezpośrednio włamań oraz nie zauważają niewielkich trzęsień ziemi.

Graf nie ma węzłów dla których 'JohnCalls' lub 'Mary Calls' byliby rodzicami (np. 'Mary słuchająca muzyki' lub 'John mylący telefon z alarmem'). To właśnie dlatego krawędzie idące z 'Alarm' do 'JohnCalls' i 'Mary Calls' mają przypisaną do siebie niepewność. Podejście takie powoduje że model staje się bardziej uniwersalny, ponieważ przez wprowadzenie niepewności bierzemy pod uwagę wszystkie czynniki losowe takie które potencjalnie mogłyby wpłynąć np. na działanie alarmu (wysoka wilgotność, awaria zasilania, rozładowany akumulator, przecięte przewody, martwa mysz utknięta w dzwonku itp.). W ten sposób mały agent może poradzić sobie z bardzo dużym światem. Stopień przybliżenia można poprawić, jeśli wprowadzimy dodatkowe istotne informacje.

4.9. Uczenie Bayesowskie

Ideą uczenia Bayesowskiego jest wyliczenie rozkładu prawdopodobieństwa parametrów wyjściowych nowego przypadku pod warunkiem jego parametrów wejściowych i wszystkich przypadków treningowych. Załóżmy, że nowy przypadek ma parametr wejściowy $X = x$ (który będziemy zapisywać jako x) i parametr wyjściowy Y. Celem będzie policzenie $P(Y|x \wedge E)$, gdzie E jest zbiorem treningowym przypadków.

4.10. Wnioskowanie probabilistyczne

Jest to obliczanie prawdopodobieństw następujących dla propozycji zapytań, biorąc pod uwagę zaobserwowane dowody.

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Figure 13.3 A full joint distribution for the *Toothache*, *Cavity*, *Catch* world.

4.10.1. Marginalizacja

Proces uzyskania rozkładu na jakiś podzbiór zmiennych albo na jedną zmienną, aby uzyskać prawdopodobieństwo marginalne, za pomocą sumowania prawdopodobieństwa dla każdej możliwej wartości innych/pozostałych zmiennych, tym samym usuwając je z równania.

np. prawdopodobieństwo marginalne próchnicy: $P(cavity) = 0.108 + 0.012 + 0.072 + 0.008$

5 Kartkówka 4 - Programowanie z ograniczeniami

Programowanie z ograniczeniami służy do rozwiązywania tzw. CSP (constraint satisfaction problem), czyli problemów spełnienia ograniczeń. Najpierw zdefiniujemy sobie trochę formalniej CSP, a następnie przejdziemy do metod służących do rozwiązywania CSP. Aby opisać CSP będą nam potrzebne pojęcia przedstawione poniżej.

5.1. Pojęcia podstawowe

5.1.1. Zmienna algebraiczna

Zmienna określa pewną cechę opisywanego przez nas świata (np. wzrost człowieka, stan włącznika itp.), zmienne zapisujemy dużymi literami, np. X, Y.

5.1.2. Dziedzina

Każda zmienna algebraiczna ma przypisaną do niej dziedzinę, którą oznaczamy $dom(X)$. Dziedzina to zbiór wartości, które może przyjmować zmienna.

Zmienne dzielimy na

- dyskretne - wartości w tej dziedzinie są skończone lub przeliczalnie skończone
- binarne - dwie wartości w dziedzinie
- ciągłe - wartości z osi liczb rzeczywistych

5.1.3. Przypisanie

Przypisanie to po prostu nadanie zmiennej wartości z jej dziedziny. Przypisanie totalne to nadanie każdej ze zmiennych wartości z jej dziedziny.
Formalnie przypisanie jest funkcją, która każdej zmiennej przypisuje wartość z jej dziedziny.

5.1.4. Ograniczenie (hard constraint)

Twarde/Silne ograniczenie (tłumaczenie własne, ang. hard constraint) lub po prostu ograniczenie ogranicza możliwe przypisania wartości do zmiennych poprzez określenie warunków, które określają czy dane przypisanie jest możliwe czy nie. Możemy mówić o ograniczeniach unarnych (na jedną zmienną), binarnych(dwie zmienne) oraz k-arnych (na wiele zmiennych).

Sposoby definiowanie ograniczeń:

- **Intension** - polega na zdefiniowaniu ograniczenia poprzez użycie symboli logicznych
- **Extension** - polega na wypisaniu wszystkich przypisań, które są poprawne

Przykład
Mamy dane zmienne $\{X, Y, Z\}$, każda z nich ma dziedzinę $\{1, 2, 3, 4\}$.
Przykładowe ograniczenie typu **intension**:

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C \leq 3)$$

Przykładowe ograniczenie typu extension:

A	B	C
2	2	4
1	1	4
1	2	3
1	2	4

5.1.5. Constraint Satisfaction Problem CSP

Teraz gdy mamy już bazę pojęć możemy sobie trochę formalniej powiedzieć czym jest CSP.

CSP Constraint Satisfaction Problem składa się z:

- zbioru zmiennych
- dziedziny dla każdej ze zmiennych
- zbioru ograniczeń

Możemy mówić o skończonych CSP oraz nieskończonych CSP, skończone CSP to takie, które posiadają skończony zbiór zmiennych i skończoną dziedzinę dla każdej ze zmiennych.

Przykład zadania z ograniczeniami

Robot w fabryce musi wykonywać określone czynności w celu wytworzenia produktu, oznaczmy je *a,b,c,d,e*. Każda z czynności może zostać wykonana w jednej z chwil czasu, które oznaczamy *1,2,3,4*. Zmiennymi w naszym zadaniu będą czasy w których robot wykonał daną czynność, i tak przez *A* będziemy oznaczać czas w jakim robot wykonał czynności *a*. Dodatkowo chcemy aby zawsze spełnione były następujące zasady:

$$\{ \quad (B \neq 3),\ (C \neq 2),\ (A \neq B),\ (B \neq C),\ (C < D),\ (A = D)\,, \\ (E < A),\ (E < B),\ (E < C),\ (E < D),\ (B \neq D)\}$$

Nasze zadanie z ograniczeniami ma więc postać:

- Zmienne: *A,B,C,D,E*

- Dziedzina:

$$dom\,(A) = \{1,2,3,4\}, \quad dom\,(B) = \{1,2,3,4\}, \quad dom\,(C) = \{1,2,3,4\}\,,$$
$$dom\,(D) = \{1,2,3,4\}, \quad dom\,(E) = \{1,2,3,4\}\,.$$

- Ograniczenia

$$\{ \quad (B \neq 3),\ (C \neq 2),\ (A \neq B),\ (B \neq C),\ (C < D),\ (A = D)\,,$$
$$(E < A),\ (E < B),\ (E < C),\ (E < D),\ (B \neq D)\}$$

5.1.6. Model dla CSP

Modelem dla CSP nazywamy takie przypisanie (czyli nadanie każdej ze zmiennych wartości z jej dziedziny), które jest zgodne ze wszystkimi ograniczeniami. Mając zadanie CSP, wiele informacji dotyczących modelu będzie dla nas ważne:

- Czy model istnieje ?
- Znalezienie modelu
- Ile jest modeli ?
- Znalezienie najlepszego modelu pod względem przyjętej metryki

W naszych dalszych rozważaniach skupimy się głównie na prezentacji metod które dadzą nam informacje o postaci modelu (o ile istnieje). Niektóre z metod pozwolą także na określenie czy model nie istnieje lub na znalezienie wszystkich możliwych modeli.

5.2. Metody rozwiązywania CSP

5.2.1. Generate-and-Test Algorithms

Nie wiem jaki będzie polski odpowiednik... przegląd zupełny (?) - W tym algorytmie chodzi o to, że sprawdzamy wszystkie możliwe przypisania po kolei, jeśli znajdziemy zgodne z ograniczeniami, to jest ono naszym znalezionym modelem. Zaletą tej metody jest oczywiście prostota, a wadą to, że jest ona możliwa do zastosowania tylko dla małych problemów. Jeśli mamy n zmiennych, które mają dziedziny o rozmiarze d, to wszystkie możliwe kombinacje to d^n.

5.2.2. Search

W metodzie Generate-and-Test przypisywaliśmy każdej zmiennej bez wyjątku wartość, a później sprawdzaliśmy to z ograniczeniami. Zastanówmy się jednak czy jest to zawsze konieczne. Jeśli wiemy, że przypisanie danej wartości do danej zmiennej narusza któreś z ograniczeń, to nie ma sensu przypisywać wartości do innych zmiennych oraz badać czy spełniają one ograniczenia - nasz model i tak nie będzie poprawny. Ta prosta obserwacja jest podstawą metody Search (Przeszukiwanie (?)).

Przeszukiwanie reprezentujemy w formie grafu w którym każdy z węzłów reprezentuje przypisanie wartości do jakiejś zmiennej (poza startowym, węzeł startowy nie reprezentuje przypisania). Jak to działa:

- Będąc w danym węźle wybierz dowolną zmienną, która nie ma w nim żadnej przypisanej wartości
- Przetestuj wszystkie możliwe przypisania wartości do tej zmiennej z jej dziedziny
- Jeśli po przypisaniu do nowej zmiennej wartości, całe przypisanie spełnia ograniczenia, to tworzy ono nowy węzeł

Trudno to zrozumieć na podstawie opisu więc spójrzmy na przykład, bo w rzeczywistości algorytm działania jest bardzo prosty:

Example 4.12. *Suppose you have a very simple CSP with the variables A, B, and C, each with domain $\{1, 2, 3, 4\}$. Suppose the constraints are $A < B$ and $B < C$. A possible search tree is shown in Figure 4.1.*

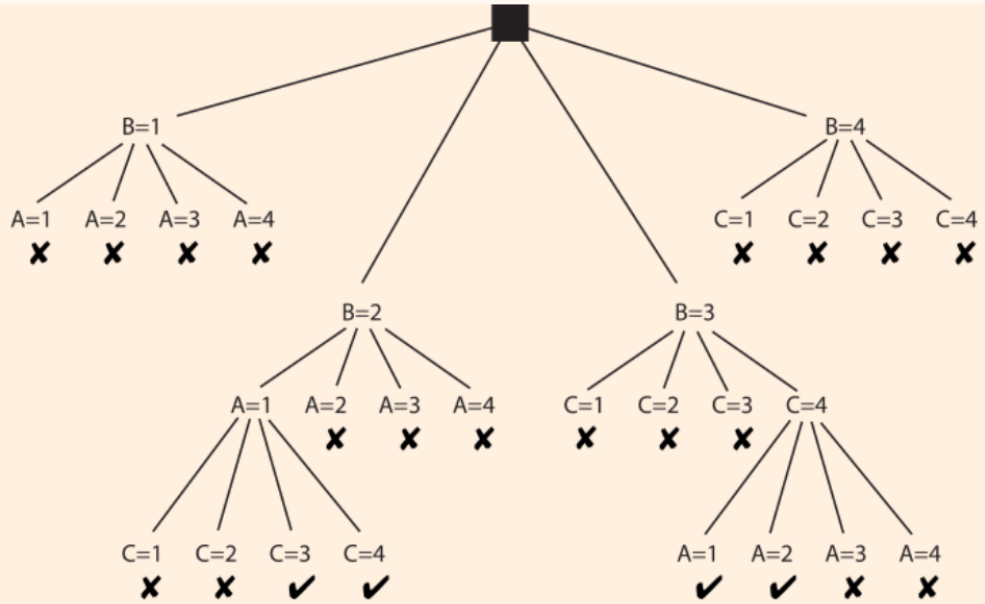


Figure 4.1: *Search tree for the CSP of Example 4.12*

Jak widzimy w drzewie interesują nas tylko końcowe węzły, to one są przypisaniem, które jest naszym modelem.

Przeszukiwanie takiego grafu np. przy pomocy algorytm depth-first search, czyli tzw. **backtracking** może być o wiele szybsze niż stosowanie metody Generate-and-Test. Metoda ta jest jednak nieco bardziej skomplikowana niż Generate-and-Test, ceną prostoty mamy szybkość działania.

5.2.3. Consistency algorithms

Consistency algorithms (algorytmy spójności (?)) polegają na badaniu dziedziny zmiennych i sprawdzaniu czy można z nich usunąć jakieś wartości, ponieważ są nieosiągalne ze względu na ograniczenia. Jak to rozumieć ? Spójrzmy na przykład:

Mamy dane zadanie z ograniczeniami w którym:

- Zmienne: *A,B*
- Dziedziny: $dom(A) = dom(B) = \{1, 2, 3, 4\}$

- Ograniczenia: $\{A < B\}$

Sprawdźmy dowolne przypisanie w którym $A = 4$, czy takie przypisanie jest w ogóle sens sprawdzać ? Łatwo można zauważyć, że nie, ponieważ $A < B$, a wiemy na podstawie dziedziny, że A oraz B są co najwyżej równe 4. Co możemy w takim razie zrobić ? Wiedząc o takiej **niespójności dziedziny** możemy usunąć z dziedziny zmiennej A wartość 4, tak aby przy przeszukiwaniu nie musieć sprawdzać wszystkich możliwości dla których $A = 4$, czyli zmieniamy dziedzinę poprzez usunięcie z niej niespójności tak aby otrzymać **spójną dziedzinę**, czyli nasza dziedzina będzie wyglądać tak: $dom(A) = \{1, 2, 3\}$. Taka jest ogólna idea algorytmu, ale jak to zalgorytmizować, bo przecież nie będziemy się opierać na spostrzeżeniach, prawda ?

W celu algorytmizacji tego algorytmu stosujemy graf nazywany siecią ograniczeń. Graf taki budujemy w następujący sposób:

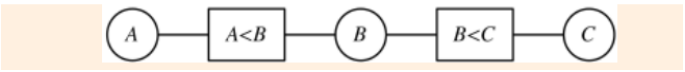
- Każdy węzeł o kształcie okręgu to zmienna
- Każdy węzeł o kształcie prostokąta to ograniczenie
- Jeśli zmienna występuje w ograniczeniu, to rysujemy krawędź nieskierowaną pomiędzy węzłem zmiennej, a węzłem ograniczenia

Przykład

Dane jest CSP w którym:

- Zmienne: A, B, C
- Ograniczenia: $\{A < B, B < C\}$
- Wartości: $dom(A) = dom(B) = dom(C) = \{1, 2, 3, 4\}$

w kroku pierwszym dla każdej ze zmiennych i każdego z ograniczeń rysujemy węzeł. W kroku drugim łączymy krawędzią zmienne z ograniczeniami, które zawierają te zmienne.



Zauważmy, że czasem mogą się zdarzyć ograniczenia tylko z jedną zmienną, przykładowo $A \neq 4$, wtedy do danego ograniczenia idzie tylko jedna krawędź z węzła A.

- **Szkic algorytmu i spójność krawędziowa** - sieć ograniczeń jest **spójną krawędziowo/gałęziowo** (arc consistent) jeśli każda z jej krawędzi jest spójna krawędziowo,
- Gałąź jest **spójna krawędziowo** jeśli dla dla każdej wartości z dziedziny zmiennej, która łączy się z węzłem ograniczenia, istnieją takie wartości innych zmiennych, które również łączą się z tym ograniczeniem, że całe ograniczenie jest spełnione. Jeśli tak nie jest, to z dziedziny możemy usunąć wartość dla której nie ma odpowiadających wartości innych zmiennych.

Algorytm ten możemy zastosować przed algorytmem Search aby wyeliminować z dziedziny niepotrzebne wartości.

5.2.4. Domain splitting lub inaczej case analysis

Idea algorytmu polega na podziale dziedziny danej zmiennej na mniejsze podzbiory. Następnie problem rozwiązujemy zakładając, że zmienna znajduje się w jednym z tych podzbiorów, później, że w kolejnym itd. Jeśli znajdziemy rozwiązanie w jednym z podzbiorów, nie musimy przeglądać pozostałych. Nie trudno zauważyć, że metod jest pewną formą algorytmu Search, a więc nic nowego, ale metoda ta może też mieć ciekawe zastosowanie w połączeniu z *Consistency algorithm*:

Jedną z efektywnych dróg rozwiązania CSP jest zastosowanie spójności krawędziowej do uproszczenia sieci przed każdym krokiem dzielenia dziedziny.

5.2.5. Variable elimination (VE)

Metoda ta jest w pewnym sensie podobna do metody *Domain splitting*. W metodzie *Domain splitting* usuwaliśmy z dziedziny wartości dla danej zmiennej, z kolei w *Variable elimination* usuwany całą zmienną z sieci.

Ogólna idea:

Metoda ta działa przez usunięcie z sieci zmiennej i zastąpienie tej zmiennej ograniczeniami nałożonymi na inne zmienne, które były powiązane ograniczeniami z usuwaną zmienną. Usuwana zmienna oraz wszystkie ograniczenia w której się zawiera są usuwane z sieci, za to pojawiają się w sieci nowe ograniczenia, których dotąd nie było, te nowe ograniczenia odzwierciedlają wpływa usuniętej zmiennej na sieć.

Ogólny schemat działania:

Metoda ta działa poprzez tworzenia łączeń (join) na możliwych wartościach zmiennych. Najpierw wypisujemy wszystkie możliwe kombinacje zmiennych dla usuwanej zmiennej oraz innych, które są z nią w jakieś relacji przez ograniczenia. Później robimy join przez wartości tej usuwanej zmiennej. Na koniec robimy join przez wartości zmiennych z ograniczeń. Wszystko to brzmi bardzo abstrakcyjnie więc zobaczmy to na przykładzie:

Example 4.22. *Consider a CSP that contains the variables A, B, and C, each with domain $\{1, 2, 3, 4\}$. Suppose the constraints that involve B are $A < B$ and $B < C$. There may be many other variables, but if B does not have any constraints in common with these variables, eliminating B will not impose any new constraints on these other variables. To remove B, first join on the relations that involve B:*

A	B
1	2
1	3
1	4
2	3
2	4
3	4

$\bowtie$

B	C
1	2
1	3
1	4
2	3
2	4
3	4

$=$

A	B	C
1	2	3
1	2	4
1	3	4
2	3	4

To get the relation on A and C induced by B, project this join onto A and C, which gives

A	C
1	3
1	4
2	4

This relation on A and C contains all of the information about the constraints on B that affect the solutions of the rest of the network.

The original constraints on B are replaced with the new constraint on A and C. VE then solves the rest of the network, which is now simpler because it does not involve the variable B. To generate one or all solutions, the algorithm remembers the joined relation on A, B, C to construct a solution that involves B from a solution to the reduced network.

5.2.6. Local search

Te algorytmy przeszukują przestrzeń rozwiązań tylko w ograniczonym zakresie. Dedykowane są dla wielkich przestrzeni rozwiązań w których przeszukanie całej przestrzeni jest niemożliwe. Algorytmy te przeszukują tylko pewne fragmenty przestrzeni rozwiązań. Nie dają gwarancji, że rozwiązania zostanie znalezione nawet jeśli istnieje więc nie mogą udowodnić, że rozwiązanie nie istnieje (algorytmy, które dają gwarancje znalezienia rozwiązania jeśli istnieje nazywamy kompletnymi). Dobrze się sprawdzają w problemach w których z góry wiemy, że rozwiązanie istnieje.

Metody te działają najpierw przypisują każdej ze zmiennych jakąś wartość, a następnie dążą do poprawy przypisań iteracyjnie, poprzez wykonywanie przypisań zgodnie z przyjętą metodą generacji nowych rozwiązań. Dlatego właśnie mówimy tutaj o całej grupie algorytmów, ponieważ generacja nowych rozwiązań może być zrealizowana na wiele sposobów. Przy rozwiązywaniu danego problemu chcemy jak najlepiej dobrać metodę do rozwiązywanego zagadnienia.

My omówimy sobie dwie metody *Local search*:

- *Random sampling* - w metodzie tej na początku losowane jest losowe przypisanie totalne, a następnie w każdej iteracji całe przypisanie totalne jest losowane od nowa
- *Random walk* - w metodzie tej na początku losowane jest losowe przypisanie totalne, a następnie w każdej iteracji zmieniana jest losowa wartość tylko jeden zmiennej

Każda z tych metod może przynosić lepsze rezultaty niż druga w zależności od tego jaki jest rozkład przestrzeni rozwiązań.