Sztuczna Inteligencja

Soma Dutta

Wydział Matematyki i Informatyki, UWM w Olsztynie
soma.dutta@matman.uwm.edu.pl

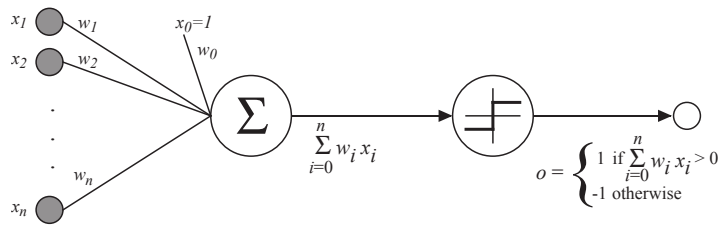
Wykład - 12: Systemy uczące się

Semestr letni 2022

Podsumowanie

Uczenie się perceptronu: jak wyuczyć perceptron dla danej funkcji, czyli jak nauczyć się wag dla jednego perceptronu. Tutaj, dokładniej problemem uczenia się jest określenie wektora wag, który powoduje że perceptron generuje poprawne wyjścia +1/0 dla każdego z podanych przykładów treningowych.

$\vec{x} = (x_1, x_2, \ldots, x_n)$ — wektor wejściowy (obiekt danych)



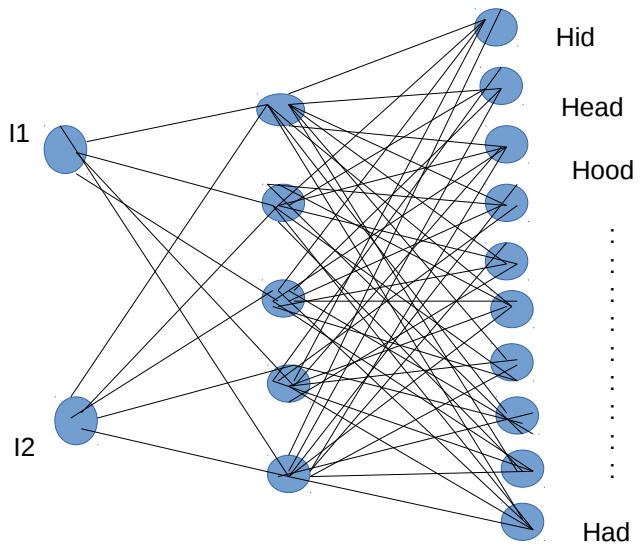
$\vec{w} = (w_0, w_1, \ldots, w_n)$ — wektor wag perceptronu
w_0 – waga przesunięcia ("przesuwa" próg funkcji aktywacji)
σ – progowa (skokowa) funkcja aktywacji perceptronu
$o(\vec{x})$ – wartość wyjścia perceptronu dla wektora $\vec{x}$

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Uczenie się jednostki liniowej na podstawie reguły delta: Regułę treningową delta najlepiej zrozumieć, biorąc pod uwagę zadanie treningu perceptronu bez progu

Wielowarstwowa sieć neuronowa

- Jak wspomniano wcześniej, pojedyncze perceptrony mogą wyrażać jedynie decyzyjne liniowe powierzchnie.
- Istnieją proste funkcje, takie jak XOR, a także złożone zadania, takie jak rozpoznawanie mowy, które obejmuje nieliniową reprezentację.
- Przedstawiona tutaj sieć została wyuczona rozpoznawania 1 z 10 dźwięków samogłoskowych występujących w kontekście 'h...d' (np. had, hid). Wejście sieci składa się z dwóch parametrów, I_1 i I_2, uzyskanych z analizy spektralnej dźwięku. Dziesięć wyjść sieci odpowiada 10 możliwym dźwiękom samogłosek. Decyzją sieci jest wyjście, którego wartość jest najwyższa.



Wielowarstwowa sieć neuronowa

Jednostki:

Jednostki podzielone są na warstwy, każda jednostka przyporządkowana jest do dokładnie jednej warstwy

Wejścia:

Wejścia podłączone są wyłącznie do jednostek znajdujących się w najniższej warstwie

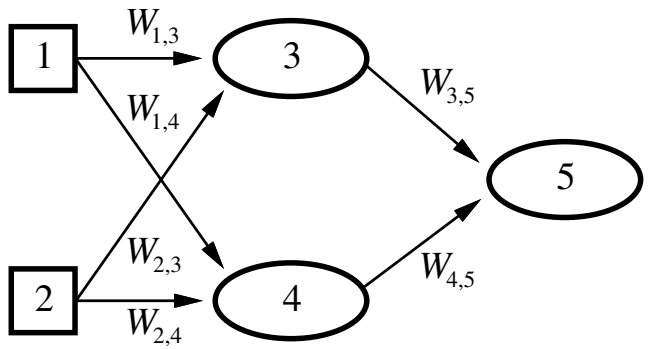
Połączenia:

Połączenia występują wyłącznie pomiędzy jednostkami z sąsiednich warstw, łączą zawsze wyjścia jednostek z warstwy niższej z wejściami do jednostek w warstwie wyższej

Wyjście:

Typowa sieć z jedną wartością funkcji ma tylko jedną jednostkę w najwyższej warstwie, wyjście z tej jednostki jest wyjściem całej sieci

Wielowarstwowa sieć neuronowa: ewaluacja



$$x_5 = \sigma(w_{3,5} \cdot x_3 + w_{4,5} \cdot x_4)$$
$$= \sigma(w_{3,5} \cdot \sigma(w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2) + w_{4,5} \cdot \sigma(w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2))$$

Uwaga:

Zastosowanie liniowych funkcji przejścia w warstwach pośrednich byłoby nieefektywne - warstwy takie można pominąć, poprzez odpowiednie przeliczenie wag i bezpośrednie połącznie warstw poprzedającej i następującej.

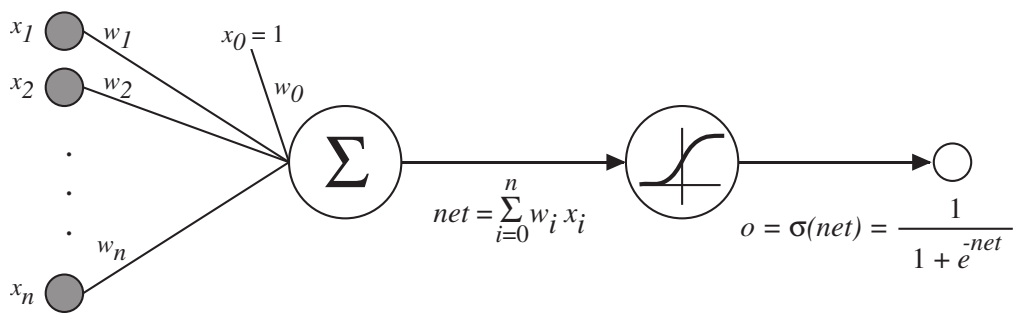
Wielowarstwowa sieć neuronowa: inna funkcja aktywacji

Jakiego rodzaju jednostki będziemy używać jako podstawy do budowy sieci wielowarstwowych?

- Wiele warstw jednostek liniowych nadal wytwarza tylko funkcje liniowe, a wolimy sieci zdolne do reprezentowania wysoce nieliniowych funkcji.
- Jednostka perceptronowa jest kolejnym możliwym wyborem, ale nieciągły próg czyni ja nieróżniczkowalną i dlatego nie nadaje się do zastosoawnia w metodzie spadku gradientu.
- Potrzebujemy jednostki, której wyjście jest nieliniową funkcją jego wejść, oraz której wyjście jest również różniczkowalną funkcji jego danych wejściowych.
- Jednym z rozwiązań jest jednostka sigmoidalna (sigmoid unit) - jednostka bardzo podobna do perceptronu, ale bazująca na wygładzonym, różniczkowalnym progu funkcjonalności.

Perceptron z sigmoidalną funkcją aktywacji

- Podobnie jak perceptron, sigmoidalna jednostka najpierw oblicza liniową kombinację danych wejściowych, a następnie stosuje próg dla wyniku. Jednak w przypadku jednostki sigmoidalnej wyjście progowe jest wyjściem ciągłym jego wejść.
- σ jest nazywany funkcją sigmoidalną lub funkcją logistyczną.
- Jego wartość wyjściowa wynosi od 0 do 1, rośnie monotonicznie wraz ze swoimi wejściami, a pochodna sigmoidalna jest łatwo wyrażalna przy użyciu jego wyjścia (konkretnie $\frac{d(\sigma(z))}{dz} = \sigma(z).(1 - \sigma(z))$).

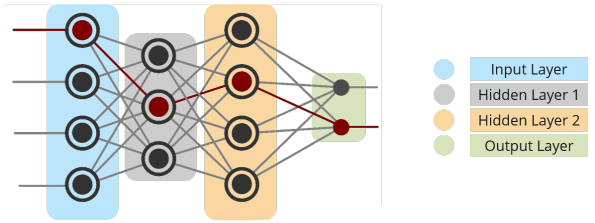


Sigmoidalna funkcja aktywacji perceptronu $\sigma(z) = \frac{1}{1+e^{-z}}$

Propagacja wsteczna: algorytm

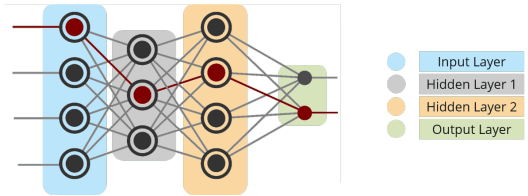
function BACK-PROP-UPDATE(*examples*, *layers*, α) **returns** a network
  inputs: *examples* - a set of examples, each with input $\vec{x}$ and output $y(\vec{x})$
    $layer_0, layer_1, \ldots, layer_n$ - neuron layers sorted from the bottom to the top
    α - the learning rate
  repeat
    for each $\vec{x} = (x_1, \ldots, x_n)$ in *examples* **do**
      for each unit $j \in layer_0$ **do**
        $o_j \leftarrow x_j$
      end for
      for each unit $j \in layer_p$ in order from $layer_1$ up to $layer_n$ **do**
        $z_j \leftarrow \sum_{i \in layer_{p-1}} w_{i,j} o_i$
        $o_j \leftarrow \sigma(z_j)$
      end for
      for each unit $j \in layer_n$ **do**
        $\delta_j \leftarrow \sigma'(z_j)(y_j(\vec{x}) - o_j)$
      end for
      for each unit $j \in layer_p$ in order from $layer_{n-1}$ down to $layer_0$ **do**
        $\delta_j \leftarrow \sigma'(z_j) \sum_{k \in layer_{p+1}} w_{j,k} \delta_k$
        $\Delta w_{j,k} \leftarrow \alpha \delta_k o_j$
        $w_{j,k} \leftarrow w_{j,k} + \Delta w_{j,k}$
      end for
    end for
  until some stopping criterion is satisfied
  return *layers* with modified weights
end function

Propagacja wsteczna: funkcja błędu



- $E(\vec{w}) = \frac{1}{2}\Sigma_{\vec{x}\in U}\Sigma_{j\in Outputs}\ (y_j(\vec{x}) - O_j(\vec{x}))^2$
  $= \frac{1}{2}\Sigma_{\vec{x}\in U}\Sigma_{j\in Outputs}\ (y_j(\vec{x}) - \sigma((\vec{w}.\vec{x})_j)^2$
- Dla każdej jednostki j w ostatniej warstwie $\delta_j = \frac{\partial Ej}{\partial \vec{w}}$ to wektor, w którym i-tej składnik $\delta_{j,i}$ jest pochodną względem i-tej składnik j-tego wektora wagi.
- $\delta_{j,i} = \frac{\partial E_j(\vec{w})}{\partial w_i} = \frac{1}{2}\Sigma_{\vec{x}\in U}\frac{\partial}{\partial w_i}(y_j(\vec{x}) - \sigma(\vec{w}.\vec{x})_j)^2$
  $= -\Sigma_{\vec{x}\in U}(y_j(\vec{x}) - \sigma(\vec{w}.\vec{x})_j).\frac{\partial\sigma(\vec{w}.\vec{x})_j}{\partial w_i}$
  Niech $\vec{w}.\vec{x} = z$. Więc $\frac{\partial\sigma(\vec{w}.\vec{x})}{\partial w_i} = \frac{\partial\sigma(z)}{\partial z}\frac{\partial(\vec{w}.\vec{x})}{\partial w_i}$ gdzie $\sigma(z) = \frac{1}{1+e^{-z}}$.
  $= -\Sigma_{\vec{x}\in U}(y_j(\vec{x}) - \sigma(\vec{w}.\vec{x})_j).\frac{\partial\sigma(z_j)}{\partial z_j}\frac{\partial(\vec{w}.\vec{x})_j}{\partial w_i}$
  $= -\Sigma_{\vec{x}\in U}(y_j(\vec{x}) - \sigma(\vec{w}.\vec{x})_j).(\sigma(z_j)(1-\sigma(z_j)))(x_i)$

Propagacja wsteczna z sigmoidalną funkcją aktywacji



Sigmoidalna funkcja aktywacji $\sigma(z) = \frac{1}{1+e^{-z}}$ we wszystkich neuronach

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \frac{1}{1+e^{-\vec{w} \cdot \vec{x}}}$$

$$\frac{\partial \sigma}{\partial z} = \left(\frac{1}{1+e^{-z}}\right)\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\frac{\partial \sigma}{\partial z}[z = \vec{w} \cdot \vec{x}] = o(\vec{x})(1 - o(\vec{x}))$$

Wartości współczynników zmiany wag δ_j

— dla neuronów j z warstwy najwyższej:

$$\delta_j \leftarrow o_j(1 - o_j)(y_j(\vec{x}) - o_j)$$

— dla neuronów j z każdej niższej warstwy p:

$$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in layer_{p+1}} w_{j,k} \delta_k$$

Propagacja wsteczna: własności

- Algorytm propagacji wstecznej działa dla dowolnego grafu skierowanego bez cykli.
- Twierdzenie: Algorytm propagacji wstecznej zbiega lokalnie do minimalnego błędu średniokwadratowego.

Uczenie indukcyjne

- Obiekty: dane reprezentujące rzeczywisty stan lub obiekt, tworzą przestrzeń obiektów X.
- Decyzja: Funkcja $dec : X \mapsto V_{dec}$ przypisująca obiektom wartość decyzji z ustalonego zbioru V_{dec}.
- Zbiór przykładów (zbiór treningowy, próbka treningowa): Ustalony zbiór obiektów z X z przypisanymi wartościami decyzji: $(x_1, dec(x_1)), (x_2, dec(x_2)), \ldots, (x_m, dec(x_m))$.
- Problem: Z danego zbioru przykładów nauczyć się funkcji (hipotezy) $h : X \mapsto V_{dec}$ aproksymującej decyzję dec tak, zeby możliwie najbardziej poprawnie przypisywała decyzyję obiektom z przestrzeni X nie występujących w zbiorze przykładów.

Drzewa decyzjna

- Indukcja drzewa decyzyjnego jest jedną z najprostszych, a jednocześnie najbardziej udanych form maszynowego uczenie się.
- Drzewo decyzyjne reprezentuje funkcję, która przyjmuje jako dane wejściowe wektor wartości atrybutów i zwraca 'decyzję' - pojedynczą wartość wyjściową.
- Wartości wejściowe i wyjściowe mogą być dyskretne lub ciągłe. Dla uproszczenia skoncentrujemy sią na problemach, w których dane wejściowe mają dyskretne wartości a dane wyjściowe mają dokładnie dwie możliwe wartości; jest to klasyfikacja boolowska (binarna), gdzie każde przykładowe dane wejściowe zostaną sklasyfikowane jako prawda (przykład pozytywny) bądź jako fałsz (przykład negatywny).
- W drzewie decyzyjnym każdy węzeł wewnętrzny odpowiada testowi wartości a_i, jednego z atrybutów. Gałęzie z węzła są oznaczone możliwymi wartościami atrybutu a_i. Każdy węzeł liścia w drzewie jest etykietowany wartością zwracaną przez funkcję jako wynik.

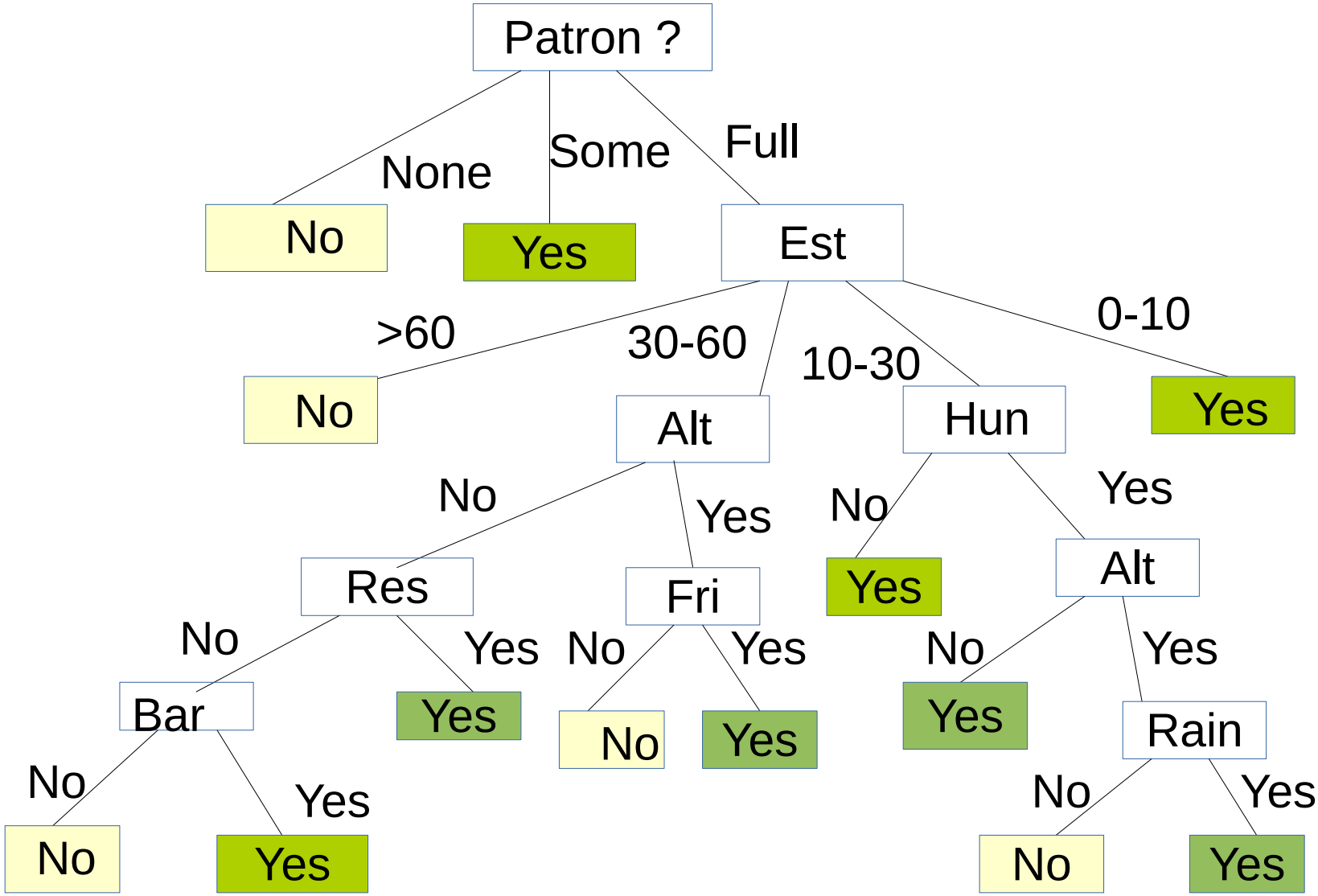
Przykład

- Predykat celowy: zdecydować, czy czekać na stolik w restauracji
- Celem jest poznanie definicji predykatu celu WillWait
- Atrybuty, na podstawie których należy podjąć decyzję:
  - Alternate: czy w pobliżu znajduje się odpowiednia alternatywna restauracja
  - Bar: czy w restauracji jest wygodny bar, w którym można czekać
  - Friday/Saturday: można czekać, jeśli jest piątek lub sobota
  - Hungry: czy jesteśmy głodni
  - Patron: ile osób jest w restauracji (brak, niektóre pełne)
  - Price: przedział cenowy dań w restauracji
  - Raining: czy na zewnątrz pada deszcz
  - Reservation: czy dokonaliśmy rezerwacji
  - Type: rodzaj restauracji (francuska, włoska, tajska lub burger)
  - WaitEstimate: czas oczekiwania oszacowany przez gospodarza (0-10 minut, 10-30, 30-60 lub > 60)

Przykład restauracji

Example	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Decision WillWait
x_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
x_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
x_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
x_4	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10-30	Yes
x_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	> 60	No
x_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
x_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
x_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
x_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	> 60	No
x_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No
x_{11}	No	No	No	No	None	$	No	No	Thai	0-10	No
x_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Drzewo decyzyjne



Ekspresyjność drzewa decyzyjnego

- Drzewo decyzyjne jest logicznie równoważne z twierdzeniem, że atrybut celowy lub atrybut decyzyjny jest prawdziwy tylko wtedy, gdy atrybuty wejściowe spełniają jedną ze ścieżek prowadzących do węzła liścia o wartości prawdziwy.

  $$Goal \Leftrightarrow Path_1 \vee Path_2 \vee \ldots \vee Path_n$$

  gdzie każda ścieżka jest koniunkcją testów atrybut-wartość wymaganych do podążenia tą ścieżką.
- Na przykład: $Path_1 \equiv (Patron = Full) \wedge (WaitEstimate = 0 - 10)$.
  $Path_2 \equiv$
  $(Patron = Full) \wedge (WaitEstimate = 10 - 30) \wedge (Hungry = No)$.
- Zatem całe wyrażenie jest równoważne DNF.

Indukowanie drzewa decyzyjnego na podstawie przykładów

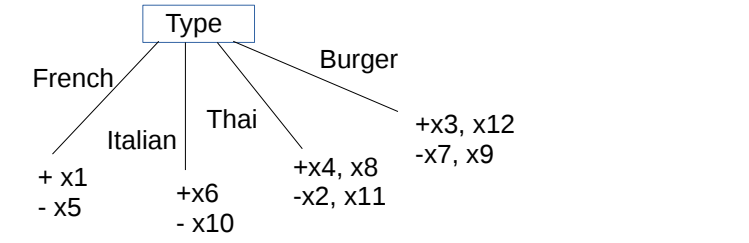
- Przykład systemu decyzyjnego ma postać $(x, dec(x))$, gdzie x jest wektorem wartości atrybutów wejściowych, a $dec(x)$ jest jedną z wartości atrybutu decyzyjnego.
- Jeśli mamy n atrybutów i każdy atrybut ma dwie możliwe wartości, drzewa decyzyjne to funkcje od zbioru z 2^n do zbioru wartości z 2 elementami. Istnieje więc 2^{2^n} możliwych drzew decyzyjnych.
- Na przykład tylko z dziesięcioma atrybutami boolowskimi naszego problemu w restauracji do wyboru jest 2^{1024} różnych funkcji.
- Musimy więc szukać dobrej hipotezy na tak dużej przestrzeni.
- Zatem celem jest skonstrukowanie drzewa, które jest zgodne z pozytywnymi przykładami (tzn. $dec(x) = prawda$) i jest tak małego rozmiaru, jak to możliwe.

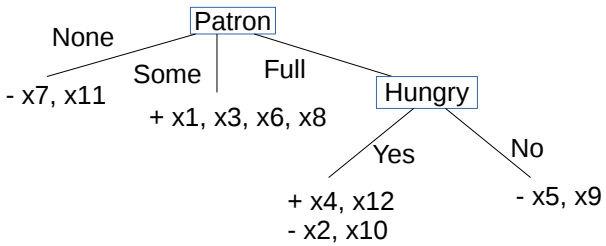
Znajdowanie drzewa decyzyjnego

- Pozytywne przykłady: te, dla których WillWait ma wartość 'Tak' (np. $x_1, x_3, \ldots$)
- Przykłady negatywne: te, dla których WillWait ma wartość 'Nie' (np. $x_2, x_5, \ldots$)
- Szukamy drzewa, które jest zgodne z przykładami i jest tak możliwie małego rozmiaru. Niestety znalezienie najmniejszego spójnego z przykładami drzewo jest trudnym problemem; nie ma sposobu na skuteczne przeszukanie 2^{2^n} drzew.
- Za pomocą prostych heurystyk możemy jednak znaleźć dobre przybliżone rozwiązanie: małe (ale nie najmniejsze) niesptrzeczne z przykładami drzewo.
- Algorytm DECISION-TREE-LEARNING bazuje na strategii zachłannej dziel i rządź: zawsze najpierw przetestuj najważniejszy atrybut. Ten test dzieli problem na mniejsze podproblemy, które można następnie rozwiązać rekurencyjnie.
- 'Najważniejszy atrybut' to ten, który najbardziej wpływa na klasyfikację przykładu.

Zły atrybut i znaczący atrybut

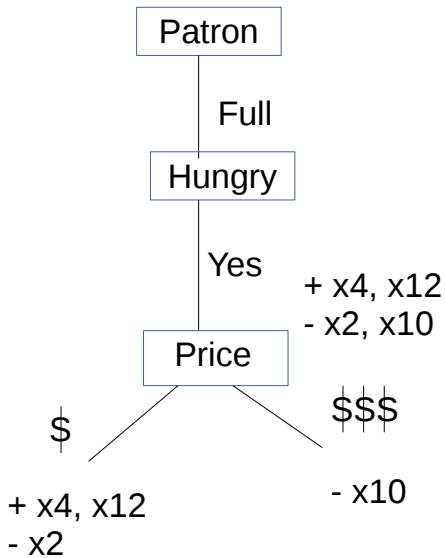
- ‘Type’ jest słabym atrybutem, ponieważ dla każdej możliwej wartości ma taką samą liczbę pozytywnych i negatywnych przykładów.



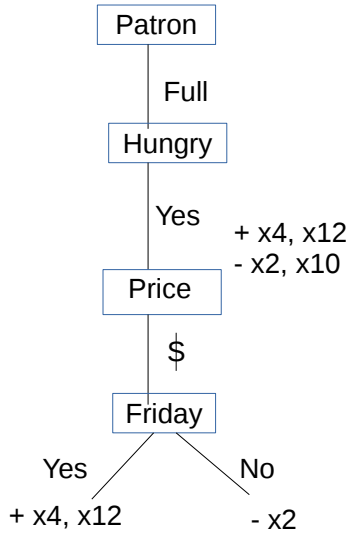


- ‘Patron’ jest bardzo istotnym atrybutem, ponieważ dla wartości ‘None’ i ‘Some’ cały zbiór danych jest podzielony na przykłady negatywne i pozytywne.
- Tak więc po pierwszym teście atrybutów, gdy przykłady zostaną podzielone, możemy przejść do testu następnego poziomu z mniejszą liczbą przykładów.
- ‘Hungry’ jest wybierany jako następny, ponieważ dla wartości ‘No’ ma tylko negatywne przykłady.

- Na tym etapie ‘Alt’ nie jest dobrym atrybutem, ponieważ w przypadku ‘Alt = Yes’, ponieważ wyznacza taką samą liczbę pozytywnych i negatywnych przykładów.
- Ale ‘Price’ jest tutaj dobrym atrybutem, ponieważ dla ‘Price = $$$’ mamy tylko negatywny przykład.



- Kolejnym znaczącym atrybutem jest ‘Friday’, ponieważ dzieli przykłady na przypadki pozytywne i negatywne.



- Więc jedną z klasyfikacji niektórych pozytywnych przykładów jest: $(Patron = Full) \wedge (Hungry = Yes) \wedge (Price = \$) \wedge (Friday = Yes)$
- Dla pozostałych pozytywnych przykładów mamy następującą klasyfikację.
  $(Patron = Some)$

Niespójne przepadki

- Jeśli po zastosowaniu tej strategii dziel i rządź nie pozostanie żaden atrybut, ale nadal pozostaną pewne pozytywne i negatywne przykłady, oznacza to, że przykłady te mają ten sam opis, ale inną klasyfikację. Oznacza to, że istnieją niespójne (sprzeczne) przypadki.
- Może się to zdarzyć z powodu błędu lub szumu w danych; lub ponieważ nie możemy zaobserwować atrybutu, który odróżniałby przypadki pozytywne i negatywne.

Algorytm uczenia się drzewa decyzyjnego

function DECISION-TREE-LEARNING(examples, attributes, parent examples) returns a tree

if examples is empty then return PLURALITY-VALUE(parent examples)

else if all examples have the same classification then return the classification

else if attributes is empty then return PLURALITY-VALUE(examples)

else
  $A \leftarrow argmax_{a \in attributes} IMPORTANCE(a, examples)$

  tree $\leftarrow$ a new decision tree with root test A

  for each value v_k of A do

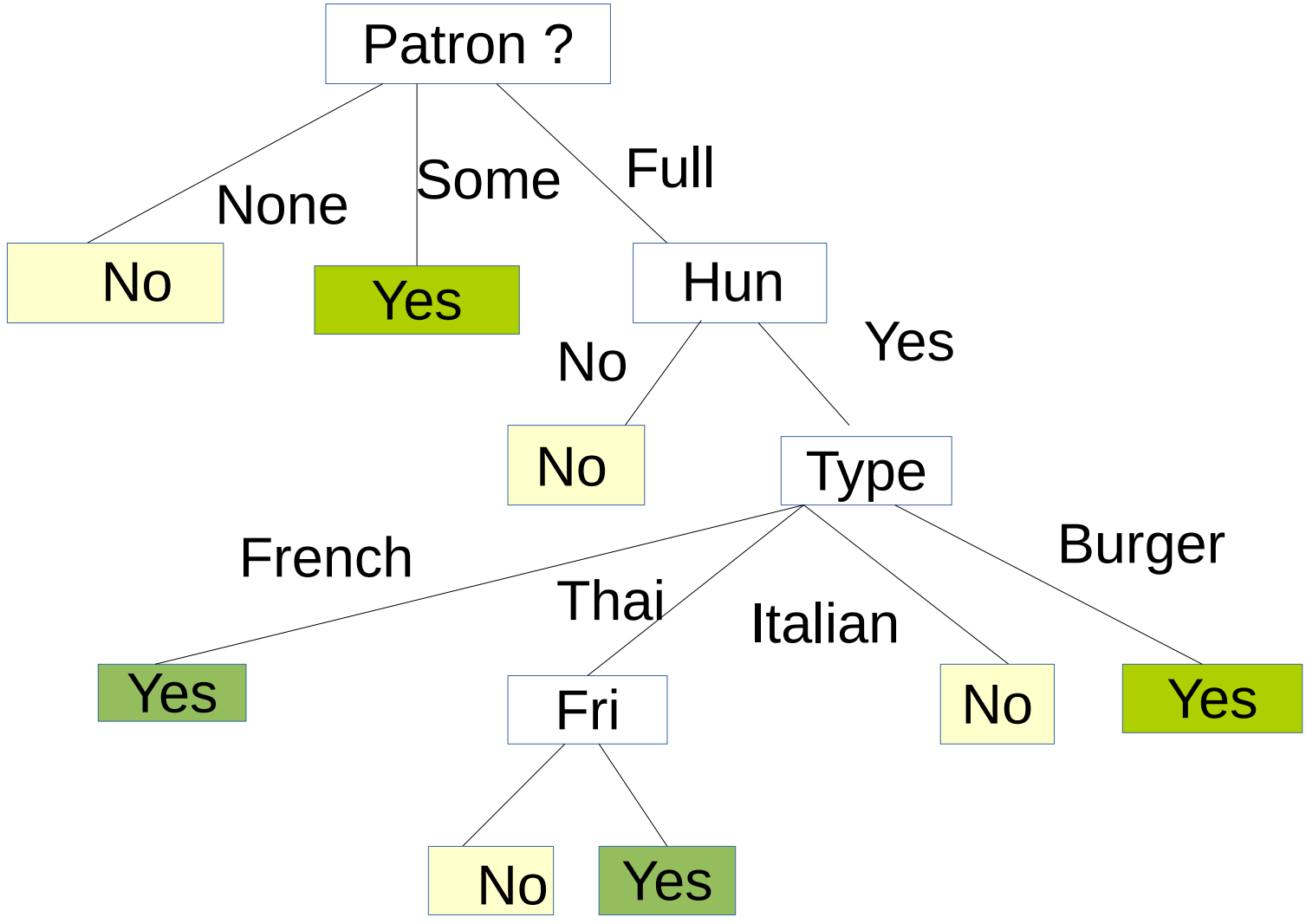
  $exs \leftarrow \{e : e \in examples,\ A(e) = v_k\}$

  subtree $\leftarrow$ DECISION-TREE-LEARNING(exs, attributes $\setminus \{A\}$, examples)

  add a branch to tree with label $(A = v_k)$ and subtree subtree

return tree

Jedno drzewo decyzyjne



- Zauważamy, że istnieje niebezpieczeństwo nadinterpretacji drzewa wybranego przez algorytm.
- Kiedy istnieje kilka zmiennych o podobnym znaczeniu, wybór między nimi jest nieco arbitralny: przy nieco innych przykładach danych wejściowych do podziału zostanie wybrana inna zmienna po pierwsze, a całe drzewo wyglądałoby zupełnie inaczej.
- Funkcja decyzyjna obliczana przez drzewo byłaby nadal podobna, ale struktura skonstrukowanego drzewa może się znacznie różnić.

Entropia

- Algorytm uczenia się drzewa decyzyjnego polega na wybraniu ważnego podzbioru atrybutów w stosunku do podanego zbioru przykładów.
- Potrzebna jest więc formalna miara ‘dość dobrych’ i ‘bezużytecznych’ atrybutów.
- Tutaj wprowadzono funkcję o nazwie ‘Ważność’ (Importance). W tym celu stosuje się pojęcie wzmocnienia informacji (information gain), które definiuje się przez entropią.
- Warto wiedzieć, że entropia jest podstawową wielkością w teorii informacji (Shanon and Weaver 1949)

Entropia

- Entropia jest miarą niepewności zmiennej losowej; pozyskiwanie informacji odpowiada zmniejszeniu entropii.
- Ogólnie entropia zmiennej losowej V o wartościach v_k, każda z prawdopodobieństwem $p(v_k)$, jest zdefiniowana jako:

  Entropy: $H(V) = \Sigma_k P(v_k) log_2 \frac{1}{P(v_k)} = -\Sigma_k P(v_k) log_2 P(v_k)$.
- Shanon zdefiniował pojęcie zawartości informacji w zmiennej w odniesieniu do jego prawdopodobieśstwa p, oznaczonego jako $I(p)$, przez pewne aksjomaty, a później odkrył, że $I(p) = log_2(\frac{1}{p})$ jest dobrym kandydatem na funkcję I. Relacja między entropią a treścią informacyjną jest podana jako $H(V) = E(I(prob(V)))$.
- Więc entropia rzutu monetą:
  $H(fair)$ = - $(0.5 log_2(0.5) + 0.5 log_2(0.5)) = 1$
- Entropia boolowskiej zmiennej losowej z prawdopodobieństwem q dla 'true' wynosi:
  $B(q) = -(q log_2 q + (1-q) log_2(1-q))$

Entropia atrybutu

- Jeśli zbiór treningowy zawiera p przykładów pozytywnych i n przykładów negatywnych, wówczas entropia atrybutu decyzyjnego dla całego zbioru to: $H(decision) = B(\frac{p}{p+n})$.
- W przykładzie związanym z restauracją $p = n = 6$. Więc odpowiednia entropia jest $B(0,5)$, która jest dokładnie 1 bit.
- Atrybut A z odrębnymi wartościami dzieli zbiór treningowy na podzbiory $E_1, E_2, \ldots E_d$. Każdy podzbiór E_k ma p_k pozytywne przykłady i n_k negatywne przykłady. Tak więc, jeśli pójdziemy wzdłuż gałęzi, potrzebujemy dodatkowych $B(\frac{p_k}{p_k+n_k})$ bitów informacji, aby odpowiedzieć na pytanie.
- Losowo wybrany przykład z zbioru treningowego ma k-tą wartość dla A z prawdopodobieństwem $\frac{p_k+n_k}{p+n}$.
- Tak więc oczekiwana entropia pozostała po przetestowaniu atrybutu A wynosi: $Remainder(A) = \Sigma_{k=1}^{d} \frac{p_k+n_k}{p+n} . B(\frac{p_k}{p_k+n_k})$
- Uzyskanie informacji (information gain) z testu atrybutu A to oczekiwane zmniejszenie entropii: $Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$

Zysk informacji definiujący funkcję IMPORTANCE

- $Gain(A)$ jest tym, czego potrzebujemy do realizacji funkcji IMPORTANCE.
- Na przykład:
  $Gain(Patron) = B(\frac{6}{12}) - Remainder(Patron)$
  $= 1 - [\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})] \approx 0.541$ bits
  $Gain(Type) = 1 - [\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{2}{4}) + \frac{4}{12}B(\frac{2}{4})] = 0$ bit
- To potwierdza, że Patron jest lepszym atrybutem do podziału.
- W rzeczywistości w podanym przykładzie 'Patron' ma maksymalne wzmocnienie dowolnego z atrybutów i będzie wybierany przez algorytm uczenia decyzji jako węzeł początkowy.

Dziękuję za uwagę