

Autostopem przez gALAIktykę: Intuicyjne omówienie zagadnień

TOM II: UCZENIE MASZYNOWE

Nie panikuj!

Autorzy: **Iwo Błądek**
Konrad Miazga

Oświadczamy, że w trakcie produkcji tego tutoriala nie zginęły żadne zwierzęta, nie została obrażona żadna opcja polityczna i nie stwierdzono ataku filozoficznych zombie.

1 Wprowadzenie

STUDENCIE lub studentko kognitywistyki, przeznaczenie chciało, żebyś uczestniczył/a w zajęciach ze Sztucznej Inteligencji i Sztucznego Życia. Przedmiot to zacy i gęboki, acz dla nieprzygotowanego umysłu ciężki do zgłębienia. Mając tedy powyższe na uwadze, ten dokument w Twoje ręce oddajemy. Staraliśmy się w nim, w sposób możliwie intuicyjny i prosty, przedstawić najważniejsze idee uczenia maszynowego dotyczące. Również często popełniane błędy tu przedstawiliśmy, bo uchronić Cię przed tego typu potknięciami na zaliczeniach czy też egzaminie.

1.1 Prawa autorskie i tym podobne

Niektóre obrazki w tym dokumencie pochodzą z <http://www.freepik.com>.

2 Uczenie maszynowe – informacje wstępne

Uczenie maszynowe i **sztuczna inteligencja** to blisko powiązane ze sobą dziedziny. W obu z nich cel jest podobny: wytworzenie „czegoś”TM, co będzie potrafiło zachowywać się inteligentnie. Jak wiemy, inteligencja nie jest prosta do zdefiniowania. Niektórzy twierdzą, że ludzie to istoty inteligentne. Inni twierdzą, że w ogólności nie.

Kiedy stworzono komputery szybko zauważono, że potrafią realizować pewnego rodzaju obliczenia szybciej niż ludzie. Tak w ogóle samo słowo 'komputer' było kiedyś nazwą na człowieka, którego praca polegała na rutynowym wykonywaniu obliczeń. Skoro więc maszyny potrafiły przeprowadzać obliczenia albo grać w gry i wygrywać z ludźmi, to może po ulepszeniach będą też potrafiły tworzyć sztukę albo rozważać bezsens życia? Może ludzie to tak naprawdę tylko bardzo skomplikowane maszyny?






Sztuczna inteligencja miała stworzyć maszyny dorównujące ludziom pod każdym względem, jednak pomimo ogromnego początkowego entuzjazmu do dzisiaj nie udało się takich maszyn stworzyć. Uzyskano jednak po drodze bardzo dużo ciekawych wyników. Na przykład programy, które potrafią znajdować regularności w ogromnych zbiorach danych, albo programy zdolne do przeprowadzania dedukcji.

Niezupełnie formalny podział jest taki, że metody „symboliczne”, często mające duży związek z logiką, zaliczane są do **sztucznej inteligencji**, a te oparte przede wszystkim na statystyce do **uczenia maszynowego**.

2.1 Zadanie klasyfikacji

Zanim przejdziemy dalej musimy powiedzieć coś o tym, jakie konkretnie zadanie będzie realizować większość algorytmów uczenia maszynowego poznawanych na zajęciach. Jest to **zadanie klasyfikacji**. Idea jest naprawdę bardzo prosta.

O klasyfikacji możemy myśleć jak o łączeniu pewnych elementów w pary. Robimy tak bardzo często w naszym życiu. Może na przykładzie:

	Jedzenie
	Jedzenie
	Picie
	Picie
	Jedzenie

Do każdego produktu przypisaliśmy pewną etykietę mówiącą o tym, czy produkt jest do jedzenia czy do picia. W uczeniu maszynowym taką etykietę nazywa się **klasą decyzyjną**. W tym problemie mamy dwie klasy decyzyjne: 'Jedzenie' i 'Picie'.

Póki co w naszym zadaniu klasyfikacji brakuje wyzwania. Mamy obrazki produktów i wiemy odgórnie, co jest czym. Założmy jednak, że pojawia się kolejny element, który widzimy po raz pierwszy w życiu:



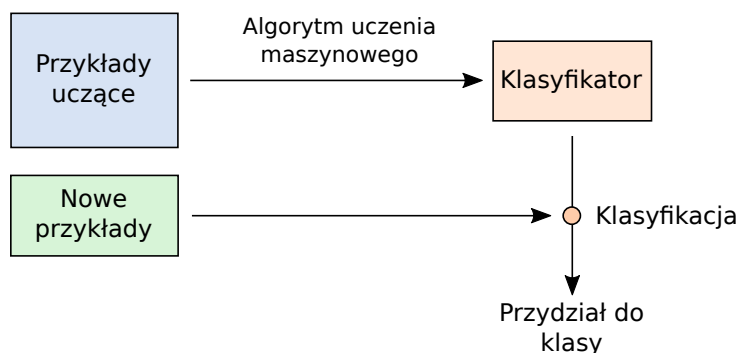
Nie widzieliśmy wcześniej dokładnie takiego samego. Czy jesteśmy na przegranej pozycji i nie posiadamy żadnych przesłanek by twierdzić, że to jedzenie albo picie? Niezupełnie. Mamy przecież powyżej naszą listę produktów i ich *klas decyzyjnych*. Spróbujmy się dowiedzieć, jakie

cechy ma 'Picie', a jakie 'Jedzenie'. W tym wypadku łatwo stwierdzić, że 'Picie' generalnie jest wyższe niż szersze i jakoś tak okrągłe kształty występują w nim częściej niż w jedzeniu.

Taki proces rozumowania nazywa się **indukcją** (powinno brzmieć znajomo...). Nie posiadamy żadnej gwarancji, że dobrze zgadniemy, jednak jak już musimy strzelać to raczej w to, co daje nam wyższe prawdopodobieństwo sukcesu. Algorytmy uczenia maszynowego indukują więc pewną wiedzę z podanego im **zbioru przykładów**, by móc następnie klasyfikować, czyli przypisywać etykiety (*klasy decyzyjne*) elementom spoza tego zbioru.

2.2 Klasyfikator

Poniższy diagram przedstawia sposób wykorzystania algorytmu uczenia maszynowego:



Jak widać, algorytm uczenia maszynowego (górną strzałką) działa na pewnym zbiorze **przykładów uczących**. Widzieliśmy już czym są te przykłady w poprzedniej podsekcji i mam nadzieję czujemy, że to nic skomplikowanego. Skomplikowane rzeczy przyjdą z czasem.

Algorytmy uczenia maszynowego produkują coś, co nazywane jest **klasyfikatorem**. Klasyfikator to taki program komputerowy, który dostając jakiś wcześniej nieznaną przykład potrafi przypisać mu klasę decyzyjną (etykietę), czyli mówimy, że potrafi go **zaklasyfikować**. W poprzedniej podsekcji to właśnie odpowiedni klasyfikator niejawnie posłużył nam do nadania etykiety szklance z jakimś płynem (oczywiście bezalkoholowym).

2.3 Przykłady uczące

Rozumiemy już mniej więcej ideę stojącą za przykładami uczącymi. Musimy jednak powiedzieć o nich trochę więcej, ponieważ poprzednio pokazany przykład był trochę uproszczony. Uproszczenie to polegało na tym, że nie rozróżnialiśmy bezpośrednio **cech** obiektów. Standardowe algorytmy uczenia maszynowego potrzebują klarownie wskazanych cech, które mają brać pod uwagę – nie są tak jak my zdolne do ich automatycznego wytworzenia. Musimy im je więc podać na tacy¹.

Cechy te, nazywane formalnie **atrybutami**, są zdefiniowane dla każdego przykładu uczącego. Mogą przyjmować wartości na dowolnej ze skal opisanych w sekcji 3. Z racji tego, że każdy przykład uczący ma jakieś wartości na wszystkich atrybutach, możemy je przedstawić w tabelce. Poniższa tabelka przedstawia dość popularny zbiór danych uczących dotyczący określania tego, czy pogoda jest dobra do gry w tenisa:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	sunny	hot	high	weak	NO
D2	sunny	hot	high	strong	NO
D3	overcast	hot	high	weak	YES
...
D13	overcast	hot	normal	weak	YES
D14	rain	mild	high	strong	NO

¹Albo wykorzystać specjalizowane algorytmy ekstrakcji cech, o których nie będziemy mówić, gdyż to bardziej zaawansowane techniki mało omawiane nawet na Informatyce.

Jak widzimy, pewien „ekspert” wyróżnił pewne cechy pogody, które uważał za potencjalnie istotne. Atrybut *PlayTennis* jest specjalny, ponieważ zawiera nasze etykiety, czyli klasy decyzyjne (tutaj mamy dwie klasy decyzyjne: YES i NO). Nazywany on jest **atrybutem decyzyjnym**. Zbiór przykładów uczących ma zawsze tylko jeden taki atrybut. Atrybut *Day* jest z kolei **identyfikatorem**, czyli unikalną nazwą pojedynczego przykładu uczącego. Identyfikatory nie biorą udziału w procesie uczenia. Identyfikatory nie są też obowiązkowe, w przeciwieństwie do atrybutu decyzyjnego, który przed rozpoczęciem uczenia zawsze musi zostać wyróżniony.

2.4 Więcej o atrybutach

2.4.1 Brakujące wartości

Czasami dla jakiegoś przykładu nie znamy jego wartości na pewnym atrybucie. Może to wynikać z różnych powodów. Przykładowo, pewnego dnia mógł nam się zepsuć miernik wilgotności i nie mamy żadnego jej pomiaru na tamten dzień. Chcielibyśmy wiedzieć co robić w takich, jakże życiowych, sytuacjach.

Możliwe sposoby radzenia sobie:

- wstawienie wartości występującej najczęściej dla danego atrybutu,
- wstawienie specjalnej wartości atrybutu dla nieokreślonych przypadków (wartość 'unknown'),
- odrzucenie przypadku z brakującą wartością.

2.4.2 Dyskretyzacja

Dyskretyzacja jest przejściem z ciągłej dziedziny (np. liczby rzeczywiste) do dyskretnej (liczby całkowite lub uporządkowane etykiety). Ciągła dziedzina może przyjmować dowolne wartości pośrednie, podczas gdy dziedziny dyskretne mają wyróżnione „kroki”, pomiędzy którymi wiemy, że żadnych wartości nie ma (np. dla liczb całkowitych nie ma żadnej liczby między 1 i 2). Dyskretyzacja wiąże się z pewną utratą informacji.

Przykład 2.1 Zamiana pomiaru temperatury w °C na jakościowe określenia, takie jak: 'hot', 'mild', 'cold'. Możemy na przykład uznać, że wartości $< 15^{\circ}\text{C}$ to 'cold', następnie $< 30^{\circ}\text{C}$ to 'mild', a $\geq 30^{\circ}\text{C}$ to 'hot'.

Przykład 2.2 Zaokrąglanie do liczb całkowitych. Przykładowo: $25.5 \rightarrow 26$, $10.2 \rightarrow 10$, $31.9 \rightarrow 32$.

2.5 Rodzaje algorytmów uczenia maszynowego

Algorytmy uczenia maszynowego można w naturalny sposób podzielić na dwa rodzaje: **uczenie nadzorowane** i **uczenie nienadzorowane**. To rozróżnienie jest bardzo istotne.

2.5.1 Uczenie nadzorowane

Jest to taka sytuacja, w której algorytm uczenia maszynowego dostaje przykłady wraz z ich przypisaniem do klasy decyzyjnej. Dzięki temu możemy łatwo sprawdzić, jak dobry jest nasz algorytm, gdyż znamy optymalne przypisania poszczególnych przykładów uczących. Jest to najbardziej standardowy sposób uczenia i tak właściwie nasze rozważania wcześniej dotyczyły właśnie jego – o ile pamiętasz, mieliśmy klasy decyzyjne 'Jedzenie' i 'Picie' albo 'YES' i 'NO' przypisane ogólnie do przykładów uczących.

Nazwa wzięła się stąd, że możemy sobie wyobrazić, że to pewnego rodzaju „nauczyciel” podaje właściwe przypisania do klas. Nie wnikając w ontologiczny status nauczyciela, ktoś/coś nadzoruje naszym uczeniem podając poprawne odpowiedzi dla przykładów uczących. Zazwyczaj algorytm uczenia maszynowego dostaje je od jakiegoś eksperta albo znamy je z przeszłości.

2.5.2 Uczenie nienadzorowane

Tutaj mamy nieco odmienną sytuację. Mamy różne przykłady uczące, ale nie dysponujemy ich przypisaniem do klasy decyzyjnej. Naszym celem będzie de facto właśnie wytworzenie „klas decyzyjnych”, nazywanych **skupieniami** (ew. klastrami), przez odpowiednie pogrupowanie obiektów. Obiekty musimy pogrupować na podstawie ich wzajemnego podobieństwa. Jest to zasadniczo jedyna informacja, której możemy do tego celu użyć. Zadaniem realizowanym przez uczenie nienadzorowane nie jest więc klasyfikacja, a **grupowanie**.

Nasz zbiór przykładów uczących wygląda następująco (zwróć uwagę na brak przypisanych etykiet):



W wyniku działania algorytmu uczenia nienadzorowanego może powstać na przykład taki podział na grupy:



Kolor jest tu oznaczeniem danej grupy (skupienia) przykładów zwróconej przez algorytm. Przykłady te są dość podobne w obrębie danej grupy i raczej się dość mocno różnią od przykładów z innych grup.

Algorytmy uczenia nienadzorowanego są w pewnym sensie wyjątkowe, gdyż nie tworzą żadnego klasyfikatora. Po prostu zadanie jest inne – chcemy tak naprawdę sprawdzić, które elementy są podobne do innych. Następnie ekspert może spojrzeć na podział zaproponowany przez algorytm i nadać interpretacje różnym grupom (np. stwierdzić, że niebieska grupa powyżej dotyczy napojów, a czerwona jedzenia w kształcie trójkątów).

2.5.3 Uwagi końcowe

Poniższa tabelka przedstawia nazwy zadań realizowanych przez poszczególne rodzaje uczenia oraz przykładowe algorytmy.

Rodzaj uczenia	Zadanie	Przykłady
uczenie nadzorowane	klasyfikacja, regresja ²	ZeroR, OneR, k-NN, drzewa decyzyjne, klasyfikator Bayesa, sztuczne sieci neuronowe (backpropagation)
uczenie nienadzorowane	grupowanie, odkrywanie asocjacji ²	k-means, k-medoids, Apriori

²Zadanie regresji zostanie omówione w sekcji 9, a zadanie odkrywania asocjacji w sekcji 11.

3 Skale pomiarowe

Skala jest pewnym sposobem interpretacji pomiarów/danych. Możemy o niej myśleć jako o dodatkowej (meta-)informacji, która opisuje wzajemne relacje i możliwe operacje między wartościami w niej wyrażonymi. Omówimy różne rodzaje skal na przykładzie.

3.1 Przykład i omówienie

Scenariusz: dokonujemy pomiarów temperatury w ciągu pięciu kolejnych dni. Za każdym razem notujemy odczyty czterech różnych termometrów, z których *przypadkiem* każdy daje wyniki na skali innego rodzaju.

(Termometr 1) Skala nominalna

Zanotowane pomiary: 'jakaśA', 'jakaśB', 'jakaśC', 'jakaśD', 'jakaśD'

Jak widzimy, termometr nie zaszalał z podawaną nam informacją. Potrafimy tylko rozpoznać, kiedy temperatury są różne, a kiedy równe. Nie mamy pojęcia, jak ma się 'jakaśC' do 'jakaśD'. Innym przykładem skali nominalnej może być płeć ('K' i 'M'). Wartości na takiej skali to w zasadzie etykiety, nawet jeżeli wyglądają jak liczby (gdyby ten termometr zwracał zamiast 'jakaśX' „liczby”, i zwracałby '1', '2', '3', '4', '4', to mając wiedzę o nominalności wiemy, że twierdzenie o wyższości odczytu jednego z pomiarów nad innym jest nieuprawnione).

(Termometr 2) Skala porządkowa

Zanotowane pomiary: 'bardzo wysoka', 'średnia', 'niska', 'wysoka', 'wysoka'

Drugi termometr jest nieco lepszy. Wiemy, jak się mają temperatury między poszczególnymi dniami, to znaczy czy były wyższe czy niższe. Mamy określony z góry porządek ('bardzo niska', 'niska', 'średnia', 'wysoka', 'bardzo wysoka'). W skali nominalnej takiego porządku nie było. Nie mamy jednak niestety pojęcia, o ile dokładnie *wysoka* temperatura jest wyższa od *średniej*.

(Termometr 3) Skala przedziałowa

Zanotowane pomiary: 40°C, 20°C, 10°C, 30°C, 30°C

Z takiego termometru korzysta większość z nas. Skala przedziałowa (zwana też interwałową) pozwala, poza uporządkowaniem pomiarów, obliczać również różnice między nimi. Niby mały dodatek w stosunku do skali porządkowej, a cieszy...

(Termometr 4) Skala ilorazowa

Zanotowane pomiary: 313 K, 293 K, 283 K, 303 K, 303 K

Jednostką temperatury rozpatrywaną w fizyce nie są °C a Kelviny (K). 0°C to w przybliżeniu 273 K. Kelviny mają pewną bardzo dobrą własność: przy braku *jakiegokolwiek* ciepła obiekt ma temperaturę 0 Kelvinów. Nie można mieć niższej temperatury³. Nazywamy ją *zerem bezwzględnym*. Na każdej skali ilorazowej możemy wyróżnić jakieś zero bezwzględne.

Skąd się zatem wzięła nazwa *skala ilorazowa*? Mianowicie możemy przy zerze bezwzględnym zacząć mówić o tym, że coś jest ileś tam razy większe od innego. Jak coś ma 1 K, a coś innego ma 20 K, to jest dwadzieścia razy bardziej ciepłe. Przy skali wyłącznie przedziałowej to nie ma sensu. Jak mamy temperaturę 20°C i 40°C, to ta druga w rzeczywistości wcale nie jest 2 razy większa (czy potrafisz powiedzieć, dlaczego?). A są jeszcze w skali Celsjusza temperatury ujemne...

³ Aczkolwiek fizycy umówili się, by w pewnych szczególnych sytuacjach przypisywać cząsteczkom ujemną liczbę Kelvinów. Nie bierzemy tego pod uwagę.

3.2 Uwagi

Skale tworzą pewnego rodzaju hierarchię, w której dana skala ma „dostęp” do wszystkich operacji możliwych na skalach położonych niżej w hierarchii. Nietrudno zgadnąć, że na szczycie hierarchii znajduje się skala ilorazowa. Następnie jest skala przedziałowa, potem porządkowa, a na końcu nominalna.

4 Miary jakości modelu

Jak już wiemy, uczenie maszynowe służy do tworzenia (między innymi) klasyfikatorów, które mogą być następnie wykorzystywane do klasyfikacji nowych przykładów. Nie każde dwa klasyfikatory są jednak sobie równe – niektóre będą działać lepiej, podczas gdy inne gorzej. W tym rozdziale omówimy więc różne miary oceniania ich jakości.

4.1 Trafność klasyfikacji

Najprostszą miarą jakości klasyfikatora jest jego **trafność klasyfikacji**, tj. procent przypadków, w których trafnie rozpoznaje on klasę decyzyjną. Jakkolwiek najczęściej wykorzystywana, to należy mieć na uwadze, iż nie zawsze jest to najlepsza możliwa miara: wyobraźmy sobie lekarza badającego grupę osób pod względem obecności jakiejś rzadkiej choroby (np. występującej jedynie u 1% populacji). Jeżeli taki lekarz zawsze będzie mówił pacjentom, że są zdrowi, osiągnie on bardzo wysoką trafność (aż 99%)! Mimo to, nazwalibyśmy go szarlatanem – nie rozpoznał on ani jednego przypadku choroby!



Podobnie jest z klasyfikatorami: sama trafność klasyfikacji nie wystarczy aby stwierdzić, czy dobrze one działają. Poniższe miary jakości klasyfikatorów (oraz – w jednym przypadku – regresji) pozwolą nam lepiej zrozumieć i opisać jakość tego, co wyszło z drugiego końca algorytmu uczenia maszynowego.

Przykład 4.1 — Liczenie trafności klasyfikacji. Załóżmy, że lekarz poprawnie zdiagnozował 99 pacjentów, a 26 niepoprawnie. Łącznie miał on więc 125 pacjentów. Trafność klasyfikacji wynosić będzie:

$$\frac{99}{125} = 0.792 = 79.2\%$$

4.2 Macierz pomyłek

Po zakończeniu procedury testowania trafności naszego modelu zostajemy z listą przykładów testowych, gdzie dla każdego z nich znamy klasę rzeczywistą oraz klasę przewidywaną przez nasz model. Zliczając liczbę przypadków dla każdej z kombinacji tych dwóch klas (rzeczywistej i przewidywanej) możemy stworzyć tzw. **macierz pomyłek**.

	a	b	c
a	10	2	3
b	0	8	5
c	1	1	8

(a) Macierz pomyłek.

	a	¬ a
a	10	5
¬ a	1	22

(b) Macierz pomyłek dla klasy *a*.

	a	¬ a
a	TP	FN
¬ a	FP	TN

(c) Oznaczenia wartości macierzy pomyłek.

Tabela 1: W wierszach: klasa rzeczywista; w kolumnach: klasa przewidywana.

Tabela 1a przedstawia przykładową macierz pomyłek. Możemy na jej podstawie obliczyć trafność modelu: musimy zsumować wartości na głównej przekątnej (przypadki poprawnie rozpoznane) i podzielić je przez liczbę wszystkich przypadków testowych. W tym wypadku trafność wynosi $\frac{26}{38} \approx 68\%$. Możemy ponadto wyciągnąć z niej dodatkowe informacje: widzimy m.in. że klasa b często była przez nasz klasyfikator mylona z klasą c , podczas gdy nigdy nie pomyliliśmy jej z klasą a .

Tabela 1b przedstawia tę samą macierz pomyłek dla klasy a – tym razem połączyliśmy klasy b i c jako klasę $\neg a$: możesz porównać wartości w obu macierzach aby się upewnić czy dobrze rozumiesz w jaki sposób przeszliśmy z macierzy 1a do 1b.

Jeżeli mamy już macierz pomyłek dla klasy a , możemy nadać nazwy jej czterem wartościom zgodnie z tabelą 1c. W tej macierzy skupiamy się na klasie a , więc jako *positive* będziemy rozumieli rozpoznanie przykładu jako a , a jako *negative* rozpoznanie przykładu jako coś innego niż a . Rozpoznanie mogą być jednak poprawne – *true* oraz błędne – *false*. Jak już się pewnie spodziewasz, te cztery wartości będziemy nazywali odpowiednio:

TP – *true positive*,

FP – *false positive*,

FN – *false negative*,

TN – *true negative*.

4.3 TPrate, FPrate, Precision, Recall

Dla każdej z klas możemy obliczyć wiele wartości o różnych interpretacjach. Tutaj skupimy się na czterech z nich: **TPrate**, **FPrate**, **Precision** oraz **Recall**. W praktyce jednak okazuje się, iż *Recall* jest jedynie inną nazwą na *TPrate*, więc to już o jedną miarę mniej do zapamiętania! Miary te wyliczane są zawsze osobno dla poszczególnych klas (w poniższych przykładach, dla klasy a).

A więc idąc po kolei:

- TPrate(a) / Recall(a)

– **Wzór:** $\frac{\text{TP}}{\text{TP} + \text{FN}}$

– **Interpretacja:** Jaką część przypadków klasy a udało nam się poprawnie wykryć? Miejmy nadzieję, że jak najwięcej!

- FPrate(a)

– **Wzór:** $\frac{\text{FP}}{\text{FP} + \text{TN}}$

– **Interpretacja:** Jaką część przypadków negatywnych ($\neg a$) błędnie rozpozналиśmy jako klasę a ? Miejmy nadzieję, że jak najmniej!

- Precision(a)

– **Wzór:** $\frac{\text{TP}}{\text{TP} + \text{FP}}$

– **Interpretacja:** Jaka część naszych rozpoznań klasy a była poprawna (jak często rozpoznając klasę a mieliśmy rację)? Miejmy nadzieję, że jak największa!

Jeśli obliczymy powyższe miary dla klasy *chory* i powyżej wspomnianego przypadku szarlatana, możemy łatwo wykryć jego oszustwo: $TPrate(chory) = 0\%$, $FPrate(chory) = 0\%$, a $Precision(chory)$ nie jesteśmy w stanie obliczyć ponieważ nikt nie został rozpoznany jako chory (a więc byśmy musieli dzielić przez zero). Czy potrafisz wskazać, która z tych miar go demaskuje?

4.4 Kappa

Współczynnik kappa (oznaczany symbolem κ) będzie nam mówił gdzie znajduje się nasz klasyfikator na skali od losowego do idealnego. Zauważ, że w przeciwieństwie do miar takich jak np. TPrate, współczynnik kappa jest wyliczany dla całego klasyfikatora, a nie tylko jednej wybranej klasy! Jego wartość można wyrazić wzorem:

$$\kappa = \frac{p_o - p_e}{1 - p_e},$$

gdzie p_o oznacza trafność zaobserwowaną (*observed*), a p_e oznacza trafność oczekiwaną (*expected*). Trafność zaobserwowana (p_o) to po prostu trafność naszego klasyfikatora. Trafność oczekiwana (p_e) to trafność, której możemy się spodziewać po klasyfikatorze losowym. Jeśli przyjrzyś się wzorowi, możesz zauważyć iż $\kappa = 1$ gdy trafność naszego klasyfikatora wynosi 100% oraz $\kappa = 0$ gdy nasz klasyfikator jest klasyfikatorem losowym (tj. $p_o = p_e$).

Prawdopodobnie potrafisz już bez problemu obliczyć p_o , więc jedynym problemem na drodze do obliczenia współczynnika kappa jest obliczenie p_e . A więc jak je obliczyć? Najpierw wyjaśnijmy sobie co rozumiemy jako „klasyfikator losowy”.

Jako **klasyfikator losowy** będziemy rozumieć taki klasyfikator, który nie opiera swoich decyzji na wartościach atrybutów klasyfikowanego przypadku, a jedynie „strzela”. Jak można się spodziewać, w takim wypadku nie będziemy w stanie zaobserwować żadnego związku pomiędzy rzeczywistymi klasami przykładów a przewidywaniami naszego klasyfikatora: jeśli spojrzemy na odpowiedzi klasyfikatora dla przykładów różnych klas, np. A, B i C, to dla każdej z nich rozkład przewidywanych klas będzie wyglądał tak samo, np. 15% A, 50% B, 35% C. Oczywiście klasyfikator działający w ten sposób będzie w głównej mierze bezużyteczny (zauważ jednak, że algorytm ZeroR zawsze wyprodukuje właśnie tego typu klasyfikator – będzie on przypisywał przypadkowi zawsze tą samą klasę, **niezależnie od jego klasy rzeczywistej**).

Obliczenia trafności oczekiwanej (p_e) opierają się na macierzy pomyłek i ich przykład został przedstawiony poniżej:

	a	b	
a	25	35	=60
b	5	35	=40
	=30	=70	

(a) Sumujemy wartości w wierszach i kolumnach...

	a	b	
a	25	35	=0.6
b	5	35	=0.4
	=0.3	=0.7	

(b) Normalizujemy obliczone wartości dzieląc je przez sumę wartości całej macierzy. Zauważ, że wartości dla kolumn (jak i dla wierszy) sumują się teraz do 1.

	a	b	
a	$0.6 \cdot 0.3 = 0.18$	$0.6 \cdot 0.7 = 0.42$	=0.6
b	$0.4 \cdot 0.3 = 0.12$	$0.4 \cdot 0.7 = 0.28$	=0.4
	=0.3	=0.7	

(c) Obliczamy nowe wartości macierzy pomyłek mnożąc wartości dla odpowiednich kolumn i wierszy. Suma wartości w macierzy powinna wynosić teraz 1.

Tabela 2: W wierszach: klasa rzeczywista; w kolumnach: klasa przewidywana.

Voilà! Uzyskaliśmy oczekiwaną macierz pomyłek dla klasyfikatora losowego. Należy teraz obliczyć z niej trafność oczekiwaną: $p_e = 0.18 + 0.28 = 0.46$. Zauważ, że nie dzieliliśmy tutaj

sumy przekątnej przez sumę wartości macierzy, ponieważ wynosi ona 1 więc ta operacja nic by nie zmieniła.

Obliczmy jeszcze tylko $p_o = 0.6$ i możemy już wyliczać wartość współczynnika kappa dla tego klasyfikatora!

$$\kappa = \frac{0.6 - 0.46}{1 - 0.46} = 0.2593$$

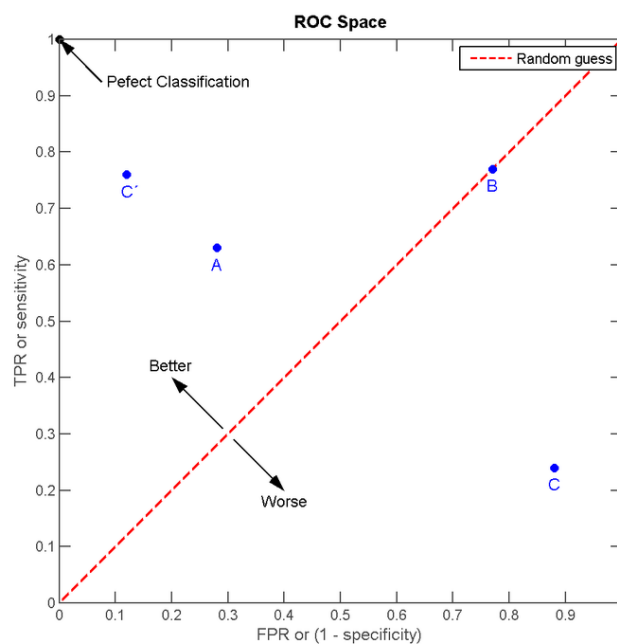
Jak widać obliczona kappa jest dość niska, a więc w tym przypadku nasz klasyfikator radzi sobie **niewiele lepiej** niż klasyfikator losowy.

4.5 Krzywa ROC, ROC area

Uwaga, poniższe rozważania uwzględniają pojęcia takie jak TPrate czy FPrate, a więc krzywa ROC oraz wartość ROC area będą zawsze obliczane dla jakiejś konkretnej klasy!

Aby wyjaśnić miarę **ROC area**, musimy najpierw wyjaśnić czym jest **krzywa ROC**. A żeby wyjaśnić krzywą ROC, musimy najpierw opowiedzieć o **przestrzeni ROC**.

Przestrzeń ROC możesz zrozumieć jako prosty wykres 2D. Na osi X będzie znajdowała się wartość miary FPrate, a na osi Y wartość miary TPrate. Jak pewnie pamiętasz, obie te miary przyjmują wartości od 0 do 1, a więc możesz sobie wyobrazić przestrzeń ROC jako swego rodzaju kwadrat. Przestrzeń ROC, razem z niektórymi interesującymi w niej punktami została przedstawiona na rysunku poniżej:



Rysunek 1: Autor: Indon

Jako że dla każdego klasyfikatora możemy wyznaczyć wartości TPrate oraz FPrate (poza nietypowymi przypadkami gdy zajdzie dzielenie przez zero – ale tym się nie przejmuj), każdy klasyfikator można przedstawić jako punkt w przestrzeni ROC o współrzędnych odpowiadającym wartościom tych miar. Jak interpretować położenie punktu odpowiadającego naszemu klasyfikatorowi w przestrzeni ROC? Aby się dowiedzieć, spójrzmy na niektóre z charakterystycznych punktów tej przestrzeni:

- **Klasyfikator idealny** – to taki klasyfikator, którego przewidywania będą zawsze zgodne z rzeczywistością. Jak możesz pamiętać, dla takiego klasyfikatora $TPrate = 1$ oraz $FPrate = 0$, a więc będzie się on znajdował w górnym lewym narożniku wykresu.

Im bliżej niego będziemy się znajdować, tym lepiej!

- **Klasyfikator losowy** – to taki klasyfikator, który „strzela”, tj. przewiduje klasę A w $p\%$ przypadków i klasę $\neg A$ w $(100 - p)\%$ przypadków. Jego miejsce na wykresie zależy od wartości p , ale zawsze znajduje się gdzieś na przekątnej – odcinku między lewym-dolnym (dla $p = 0$) a prawym-górnym (dla $p = 100$) narożnikiem przestrzeni (przykład: punkt **B**). Im dalej od niego (czyli od przekątnej) będziemy się znajdować, tym lepiej!
- **Klasyfikator antyidealny** – to taki klasyfikator, którego przewidywania nigdy nie będą zgodne z rzeczywistością. Analogicznie do klasyfikatora idealnego, zaobserwujemy dla niego $TPrate = 0$ oraz $FPrate = 1$, a więc będzie się on znajdował w prawym-dolnym narożniku wykresu.

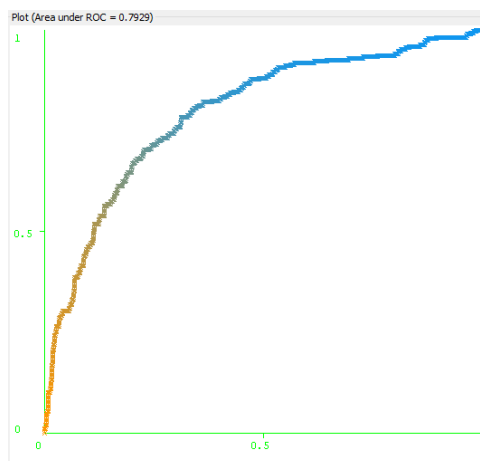
Im bliżej niego będziemy się znajdować, tym... hmmm...

Jak widzimy, chcemy się znaleźć w lewej-górnej części przestrzeni ROC – jak najbliżej narożnika, jak najdalej od przekątnej. Ale co w takim razie z prawą-dolną częścią wykresu?

Jeżeli nasz klasyfikator znajdzie się poniżej przekątnej (przykład: punkt **C**), oznacza to, iż działa on gorzej niż klasyfikator losowy – częściej będzie on pudłował niż trafiał. Jeżeli jednak jesteśmy tego świadomi, możemy nadal z niego korzystać – wystarczy zawsze klasyfikować przykłady na przekór jego wskazaniom (przykład: punkt **C'**)! Oznacza to, iż im bliżej prawego-dolnego narożnika jesteśmy, tym gorzej (bo jakość spada) ale jednocześnie tym lepiej (bo działając na przekór jakość rośnie)! W praktyce jednak twój klasyfikator powinien znajdować się **nad** przekątną (przykład: punkt **A**).

Krzywą ROC nazywać będziemy krzywą utworzoną przez połączenie punktu naszego klasyfikatora oraz wierzchołków $(0, 0)$ oraz $(1, 1)$. Mówi nam ona o tym, jakie punkty w przestrzeni ROC jesteśmy w stanie osiągnąć. Oczywiście, jesteśmy w stanie osiągnąć punkt w którym jesteśmy. Nie ma też problemu znaleźć się w wierzchołkach przestrzeni ROC. Ale co z punktami na odcinkach łączących nasze punkty? Zauważ, że jeśli co drugą klasyfikację będziemy przeprowadzać naszym klasyfikatorem (założmy że jest on w punkcie $(0.2, 0.6)$), a co drugą klasyfikatorem losowym z punktu $(0, 0)$, to taki „łączony” klasyfikator znajdzie się w punkcie $(0.1, 0.3)$ przestrzeni ROC (możesz przetestować to na kartce). A więc jesteśmy w stanie się znaleźć w pozostałych punktach naszej krzywej, łącząc odpowiednio nasze klasyfikatory.

W praktyce krzywa ROC ma jednak trochę inne zastosowanie – pozwala ona ukazać charakterystykę zachowania klasyfikatorów dla których można ustawić próg czułości. Wyobraź sobie klasyfikator który zamiast zwracać bezpośrednio decyzję: A albo $\neg A$, zwraca prawdopodobieństwo klasy A (na ile jest on przekonany, iż jest to klasa A). Zmieniając wtedy wartość progu (wartość powyżej której klasyfikator podejmie decyzję, że A), możemy generować więcej niż jeden punkt w przestrzeni ROC. Dla progu 0% uzyskamy punkt $(1, 1)$, dla progu 100% uzyskamy punkt $(0, 0)$, a dla wartości pośrednich inne punkty tworzące razem krzywą ROC. Przykładową krzywą ROC dla sieci neuronowej zamieszczono poniżej:



Rysunek 2: Krzywa ROC dla sieci neuronowej na zestawie danych *diabetes.arff*, wygenerowane za pomocą WEKI.

Problem z krzywą ROC jest taki, że jest ona stosunkowo mało zwięzłym sposobem opisu jakości klasyfikatora. W końcu każdy woli mieć jedną wartość liczbową którą można później użyć aby porównać między sobą dwa klasyfikatory, prawda? Okazuje się, iż istnieje miara bazująca na krzywej ROC która sprowadza się do jednej liczby. Jest to **ROC area**.

Nie ma tu za wiele do powiedzenia na temat ROC area – jest dokładnie tym na co wskazuje nazwa: polem powierzchni pod krzywą ROC. Zauważ, że dla klasyfikatora idealnego będzie wynosiła 1, dla losowego 0.5 a dla antyidealnego 0 – a więc im wyższa wartość tym lepiej.

4.6 Współczynnik korelacji

5 Testowanie

A więc nasz algorytm uczenia maszynowego wyprodukował jakiś klasyfikator, co teraz? Skąd możemy wiedzieć, czy będzie on dobrze działać dla zupełnie nowych danych?

Pierwsza odpowiedź, która przychodzi do głowy, to: „sprawdźmy czy jego przewidywania zgadzają się z rzeczywistością!”. Tutaj jednak pojawia się problem: jakich danych użyć do przeprowadzenia takich testów? Pierwszą myślą może być użycie tych samych przykładów na podstawie których się uczyliśmy... co zwykle będzie bardzo złym pomysłem.

Czemu? Jak temu zaradzić? Czy istnieją różne metody przeprowadzania testów? Odpowiedzi na te i inne pytania znajdziemy poniżej.

5.1 Test na zbiorze uczącym

Jedną z opcji jest pójście za pierwotnym instynktem i przetestowanie naszego modelu na danych uczących. Dlaczego jest to zły pomysł?

Wyobraź sobie drobny zbiór uczący zawierający trzy przykłady: (*pizza, food*), (*cake, food*), (*spaghetti, food*). Praktycznie każdy algorytm zauważy, iż poprawną klasą decyzyjną jest zawsze *food*, więc wszystkie nowe przykłady będą również klasyfikowane w ten sposób. Testując model na zbiorze uczącym, okaże się, że ma on 100% trafność. Hurra! Hurra?

Nie całkiem. Wyobraź sobie, że zadowoleni z rezultatów naszego klasyfikatora zaczynamy stosować go w praktyce. Nagle się okazuje, że wszystkie napoje są również niepoprawnie klasyfikowane jako jedzenie! Testy nas oszukały!

Oczywiście jest to dość drastyczny przykład, ale pokazuje główny problem z testowaniem modeli na danych uczących: będą one zawsze zwracać zawyżoną trafność. Chociaż nasz klasyfikator może być (i zazwyczaj będzie) dobrze dopasowany do danych na których się uczył, może on mieć problem z uogólnieniem wiedzy na przykłady z którymi się jeszcze nie zetknął. A jeżeli nie potrafi uogólnić wiedzy to znaczy, że w praktyce jego skuteczność będzie niska.

Można by tu wysunąć analogię do studentów uczących się na kolokwium: wykucie się odpowiedzi do pytań z poprzedniego roku niewiele pomoże, jeżeli w roku następnym pytania się zmieniają. Gdyby pytania się nie zmieniły, studenci otrzymaliby wysokie oceny sugerujące dobrą znajomość tematu. Jeżeli jednak pytania się zmieniają, dobre oceny uzyskają jedynie ci z nich, którzy rzeczywiście zrozumieli temat. W tym sensie, testowanie algorytmu na danych uczących jest niczym sprawdzanie wiedzy w oparciu o stare pytania o znanych odpowiedziach.

5.2 Test na wydzielonym zbiorze testowym

Wiemy już, że (i czemu) nie należy używać danych uczących do testowania modelu. Oczywiście więc będzie próba znalezienia innych danych na których możemy sprawdzić trafność naszego modelu. Możemy użyć osobnego pliku z danymi, albo wydzielić z danych uczących część testową. Jaka jest różnica między tymi podejściami?

Finalny klasyfikator jest zawsze uczony na pełnych danych uczących. Jeśli używamy osobnego pliku z danymi testowymi, wyniki testów będą dotyczyć dokładnie tego klasyfikatora, który został nam zaprezentowany. W przypadku wydzielenia fragmentu testowego z danych uczących, wyniki testów nie będą dotyczyły bezpośrednio klasyfikatora który został nam zaprezentowany, ale klasyfikatora nauczonego na mniejszej liczbie danych!

5.3 K-fold cross validation

Zakładając, że chcemy przetestować jakość naszego modelu ale nie posiadamy drugiego pliku z danymi, czy wydzielenie fragmentu testowego z danych uczących jest najlepszą opcją? Nie.

Poznajcie **k-fold cross validation** (znane również jako: krosvalidacja, sprawdzian krzyżowy).

Powiedzmy, że wyznaczyliśmy 33% zbioru uczącego jako dane testowe. Zauważmy, że to 33% nie zostało jeszcze użyte do nauki, a jednocześnie pozostałe 66% nie było użyte do testowania.



Rysunek 3: Wizualizacja krosvalidacji. Źródło: <http://blog-test.goldenhelix.com/wp-content/uploads/2015/04/B-fig-1.jpg>

Jest to marnotrawstwo perfekcyjnie dobrych danych, zwłaszcza jeżeli pechowo to 33% będzie zawierać szczególnie istotne przypadki z granic między klasami.

Podzielmy zbiór uczący na k równych części (ergo k -fold). Wybierzmy jedną z nich i użyjmy ją jako zbiór testowy dla modelu nauczonego na pozostałych $k - 1$ fragmentach. Powtórzmy tę procedurę $k - 1$ razy, zawsze wybierając inny fragment jako dane testowe i połączmy wyniki wszystkich testów. W ten sposób użyliśmy całego zbioru uczącego jako danych testowych, nie napotykając się jednocześnie na problemy związane z testowaniem modelu na zbiorze uczącym! Magia?



6 ZeroR

Jeżeli niektóre klasyfikatory są bardziej a inne mniej wyrafinowane, to te tworzone przez algorytm **ZeroR** zdecydowanie należą do tej drugiej kategorii. Możesz myśleć o ZeroR jak o kanapce z chlebem świata uczenia maszynowego.

Day	Outlook	Temperature	Humidity	PlayTennis
D1	sunny	hot	high	NO
D2	sunny	hot	high	NO
D3	overcast	hot	high	YES
D4	overcast	hot	normal	YES
D5	rain	mild	high	NO

(a) Rzeczywiste dane uczące.

Day	PlayTennis
D1	NO
D2	NO
D3	YES
D4	YES
D5	NO

(b) To co widzi ZeroR.

Algorytm ZeroR ignoruje wszystkie atrybuty przykładów uczących, oprócz **atrybutu decyzyjnego**. Na jego podstawie, sprawdza najczęściej występującą klasę decyzyjną i tworzy klasyfikator który zawsze będzie ją zwracać. **Algorytm ZeroR zawsze przypisuje nowe przykłady do tej samej klasy (klasy większościowej), niezależnie od wartości ich atrybutów.**

Zauważ, że taki klasyfikator nadal może mieć w niektórych przypadkach wysoką trafność: patrz na przykład lekarza szarlatana. Jednocześnie jednak, wartość współczynnika kappa będzie dla niego zawsze wynosiła zero.

7 OneR

Algorytm **OneR** jest tylko trochę bardziej wyrafinowany niż ZeroR. Podczas gdy ZeroR ignoruje wszystkie atrybuty niedecyzyjne, OneR wybiera tylko jeden z nich jednocześnie ignorując pozostałe.

W jaki sposób OneR wybiera swój „ulubiony” atrybut? Testuje trafność (**na danych uczących!**) dla klasyfikatorów zbudowanych na każdym z atrybutów osobno i wybiera ten z nich, który ją maksymalizuje.

Ok, ale w takim razie w jaki sposób OneR buduje klasyfikator na podstawie swojego wybranego atrybutu? Otóż dzieli on dane uczące na podzbiory ze względu na wartość jego „ulubionego” atrybutu. Następnie, na każdym z nich używa algorytmu ZeroR – wybiera w nich klasę większościową. Dla różnych wartości „ulubionego” atrybutu mogą występować różne klasy większościowe. Utworzony klasyfikator będzie patrzeć na wartość „ulubionego” atrybutu nowego przykładu i na jego podstawie będzie przewidywać odpowiednią klasę.

Przykład 7.1 — Tworzenie klasyfikatora algorytmem OneR. Spróbujmy prześledzić tworzenie klasyfikatora dla poniższych danych uczących:

Day	Outlook	Temperature	Humidity	PlayTennis
D1	sunny	hot	high	NO
D2	sunny	hot	high	NO
D3	overcast	hot	high	YES
D4	overcast	hot	normal	YES
D5	rain	mild	high	NO

Skonstruujmy klasyfikatory dla różnych atrybutów:

Outlook	YES	NO	trafność
sunny	0	2	5/5 = 100%
overcast	2	0	
rain	0	1	

Temperature	YES	NO	trafność
hot	2	2	3/5 = 60%
mild	0	1	

Humidity	YES	NO	trafność
high	1	3	4/5 = 80%
normal	1	0	

Jak widzimy, najwyższą trafność osiągamy korzystając z atrybutu *Outlook*. Algorytm stworzy więc następujący klasyfikator:

```
IF Outlook = sunny THEN PlayTennis = NO
IF Outlook = overcast THEN PlayTennis = YES
IF Outlook = rain THEN PlayTennis = NO
```



8 k-NN

Algorytm **k najbliższych sąsiadów**, w literaturze określane zazwyczaj jako **k-NN** (od ang. *k Nearest Neighbours*), to pierwszy bardziej skomplikowany algorytm uczenia z którym się zapoznamy. Jego główna idea to klasyfikacja poprzez analogię/podobieństwo do wcześniej poznanych przypadków. Wyobraź sobie, że jesteś lekarzem i chcesz wykorzystać uczenie maszynowe by proponowało Tobie wstępną diagnozę pacjentów. Masz do dyspozycji informacje o objawach wszystkich poprzednich pacjentów (np. rodzaj kaszlu, kataru, itd.) i ostatecznych diagnozach (np. grypa, astma). k-NN dla nowego pacjenta wyszuka dokładnie *k* najbardziej podobnych jeżeli chodzi o objawy „historycznych” pacjentów, a następnie sprawdzi, która choroba występowała wśród nich najczęściej – będzie ona ostateczną decyzją, jaką zwróci ten klasyfikator.

8.1 Miara odległości

W algorytmie k-NN kluczowe jest zdefiniowanie miary odległości między przypadkami. Posłuży ona do wyboru najbliższych sąsiadów. Dla atrybutów liczbowych stosuje się często **odległość euklidesową**. Dla dwóch przypadków *A* i *B* opisanych na *n* atrybutach odległość ta opisana jest wzorem:

$$d(A,B) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \dots + (A_n - B_n)^2},$$

gdzie X_i oznacza wartość *i*-tego atrybutu dla przypadku *X*.

Alternatywnie, wykorzystać można **odległość taksówkową** (nazywaną również *odległością Manhattan*, np. w WEKA), która sumuje tylko bezwzględne różnice między poszczególnymi wartościami:

$$d(A,B) = |A_1 - B_1| + |A_2 - B_2| + \dots + |A_n - B_n|$$

Przykład 8.1 — Liczenie odległości euklidesowej. Załóżmy, że mamy następujące przypadki:

#Example	Atr1	Atr2	Atr3	Decision
A	0	-2	6	NO
B	3	2	5	NO

Odległość euklidesową między A i B policzymy jako:

$$d(A,B) = \sqrt{(0 - 3)^2 + (-2 - 2)^2 + (6 - 5)^2} = \sqrt{(-3)^2 + (-4)^2 + 1^2} = \sqrt{9 + 16 + 1} = \sqrt{26} \approx 5.1$$

Przykład 8.2 — Liczenie odległości taksówkowej. Dla danych z Przykładu 8.1 odległość taksówkową policzymy jako:

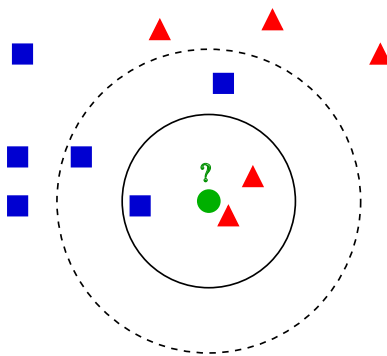
$$d(A,B) = |0 - 3| + |-2 - 2| + |6 - 5| = |-3| + |-4| + |1| = 3 + 4 + 1 = 8$$

8.2 Zarys działania

W algorytmie k-NN wszystkie przypadki uczące są po prostu zapamiętywane. To wszystko, co musi zostać zrobione w fazie uczenia.

Proces klasyfikacji nowego przypadku *X* przebiega następująco:

1. Oblicz odległość między *X* a każdym zapamiętanym przypadkiem uczącym.



Rysunek 4: Wizualizacja działania k-NN na płaszczyźnie dla różnych wartości k (3 i 5). Źródło: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

2. Posortuj odległości rosnąco i weź pierwszych k pozycji (k jest parametrem algorytmu podawanym na starcie przez użytkownika). Na pozycjach tych znajdować się będą najbliżsi sąsiedzi.
3. Wśród wybranych przypadków sprawdź przydzielone decyzje i wybierz tę, która występuje najczęściej. Jeżeli jest remis, to wybierz dowolną.

Wybrana większościowa decyzja w ostatnim punkcie stanowi ostateczny wynik zwrócony przez klasyfikator.

8.3 Pozostałe uwagi

- Parametr k zazwyczaj ustalany jest jako liczba nieparzysta, ponieważ dzięki temu maleje liczba remisów, które trzeba rozstrzygać.

8.4 Przykład 1-wymiarowy

Przedstawimy teraz k-NN na najprostszym możliwym przykładzie danych opisanych tylko jednym atrybutem. Taka sytuacja raczej nie wystąpi w praktyce, ale pozwala na bardzo czystą prezentację ogólnego działania algorytmu. Tabela z danymi wyglądać będzie następująco:

#Example	Atr1	Decision
A	1	YES
B	2	NO
C	4	YES
D	6	NO
E	8	NO
F	12	NO

Załóżmy, że dostajemy do zaklasyfikowania następujący przypadek: $(X, 3)$. Wyliczmy teraz przy pomocy odległości taksówkowej odległości między X a przykładami uczącymi. Posortujemy je też od razu rosnąco ze względu na odległość.

#	Przykład	Odległość od X	Decyzja
1.	B	$ 3 - 2 = 1$	NO
1.	C	$ 3 - 4 = 1$	YES
3.	A	$ 3 - 1 = 2$	YES
4.	D	$ 3 - 6 = 3$	NO
5.	E	$ 3 - 8 = 5$	NO
6.	F	$ 3 - 12 = 9$	NO

Widzimy więc, że wynik klasyfikacji zależy mocno od wybranej wartości k .

- Dla $k = 1$ mamy remis między B i C (mają tę samą odległość od X) i musimy arbitralnie któryś wybrać, np. losowo⁴. W takim wypadku k-NN w 50% przypadków zwróciłby YES a w 50% przypadków NO.
- Dla $k = 3$ weźmiemy dla X elementy B, C i A. Widzimy, że YES występuje wśród nich częściej, tak więc taką właśnie odpowiedź zwróci klasyfikator.
- Dla $k = 5$ weźmiemy dla X elementy B, C, A, D, E. Częściej występuje NO.
- Dla $k > 5$ będzie zawsze więcej decyzji NO.

Dla większej liczby wymiarów (atrybutów) w naszych danych jedyną różnicą w powyższej procedurze byłoby wyliczanie odległości od X.

⁴Równie dobrze można by np. próbować rozstrzygnąć taki konflikt patrząc na następnego w kolejności sąsiada. Zachowanie w takich wypadkach nie jest formalnie zdefiniowane i leży w gestii programisty.

9 Simple Linear Regression

10 k-means

Jednym z głównych zadań uczenia nienadzorowanego jest *grupowanie* podobnych elementów. W tej sekcji omówimy najbardziej podstawowy algorytm realizujący grupowanie, a mianowicie **k-means** (k-średnich). **Uważaj, by nie pomylić go z nadzorowanym algorytmem k-NN (sekcja 8)!**

W algorytmie *k-means* zakładamy, że wszystkie przykłady opisane są atrybutami liczbowymi (skale przedziałowa i ilorazowa). Tych liczbowych atrybutów może być dowolnie duża liczba, więc niech nie zmyli Cię to, że wszystko pokazywane jest na dwuwymiarowych wykresach. Wynika to tylko z łatwości wizualizacji.

10.1 Reprezentacja skupień

W algorytmie *k-means* parametr k mówi nam o liczbie tworzonych skupień (nazywanych też w literaturze grupami lub klastrami). Jedną z wad algorytmu *k-means* jest to, że liczba skupień musi zostać zadana odgórnie. Z każdym skupieniem wyznaczanym przez algorytm związany jest pewien **punkt centralny**. *Punkt centralny* intuicyjnie jest takim „sztucznym” punktem, który wyznacza środek całego skupienia. Każdy przykład ze zbioru uczącego przypisywany jest do tego skupienia, do którego punktu centralnego ma najbliżej.

Spójrz na Rysunek 5. Parametr k jest tutaj równy 2, ponieważ utworzone zostały 2 skupienia. Punkty centralne tych skupień oznaczone są trójkątami. Obserwacje (punkty A-G) oznaczone są kolorem skupienia, do którego należą. Tabelka po prawej stronie pokazuje z kolei odległości wszystkich punktów od poszczególnych punktów centralnych oraz aktualne współrzędne punktów centralnych. Można zweryfikować, że rzeczywiście punkty przydzielane są do tego skupienia, do którego centrum mają bliżej. Odległości są tutaj euklidesowe, ale w ogólności można użyć dowolnej miary odległości.

10.2 Zarys działania

Algorytm zawsze zaczyna od pewnego losowego początkowego ustawienia centrów skupień. Od tego momentu, na każdą kolejną iterację składają się dwie czynności:

1. Wyznacz aktualny przydział przykładów uczących do skupień.
2. W obrębie każdego skupienia powstałego w punkcie 1. wyznacz, poprzez uśrednienie, nowe centrum.

Algorytm kończy się, gdy w jakiejś iteracji centra skupień nie przesuną się albo gdy osiągniemy ustalony przez nas limit liczby wykonywanych iteracji.

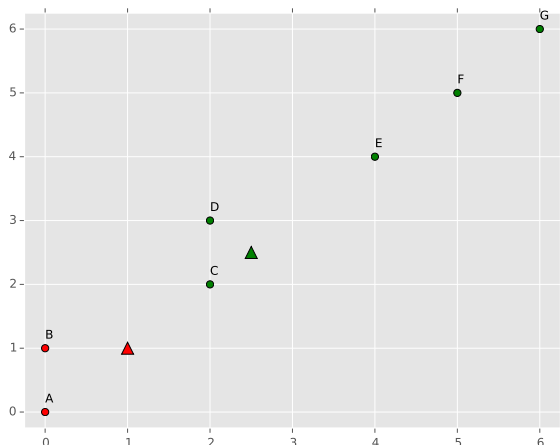
10.3 Przykład

Na Rysunkach 5-8 przedstawione są kolejne iteracje algorytmu. Możesz zweryfikować na kartce (robienie samemu jest dobrą metodą zapamiętywania), czy rzeczywiście uśrednienie punktów należących do danego skupienia prowadzi do utworzenia centrum przedstawionego na kolejnym rysunku. Punkty uśrednia się poprzez oddzielne uśrednienie ich współrzędnych.

Przykład 10.1 — Wyznaczanie punktów centralnych. Obliczymy nowe punkty centralne dla sytuacji na Rysunku 5.

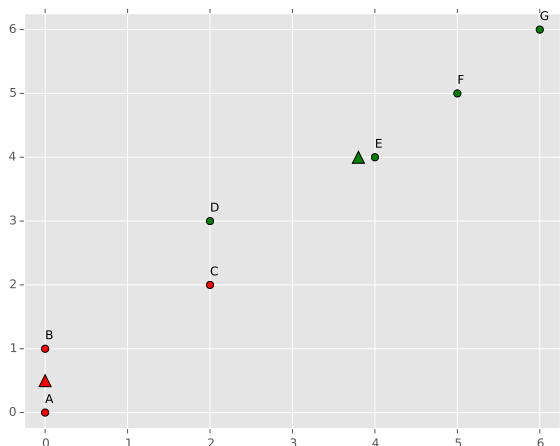
Skupienie 1 (▲): przydzielone są do niego punkty A i B.

$$\bar{x} = \frac{x_A + x_B}{2} = \frac{0 + 0}{2} = 0.0$$



punkt		odległość ▲	odległość ▲
A	(0, 0)	1.41	3.54
B	(0, 1)	1.0	2.92
C	(2, 2)	1.41	0.71
D	(2, 3)	2.24	0.71
E	(4, 4)	4.24	2.12
F	(5, 5)	5.66	3.54
G	(6, 6)	7.07	4.95
skupienie		punkt centralny	
▲		(1.0, 1.0)	
▲		(2.5, 2.5)	

Rysunek 5: Początkowe położenie centrów skupień (trójkąty: czerwony i zielony). Punkty należące do danego skupienia oznaczone są odpowiednim kolorem. W tabelce przedstawione są odległości punktów do poszczególnych centrów skupień.



punkt		odległość ▲	odległość ▲
A	(0, 0)	0.5	5.52
B	(0, 1)	0.5	4.84
C	(2, 2)	2.5	2.69
D	(2, 3)	3.2	2.06
E	(4, 4)	5.32	0.2
F	(5, 5)	6.73	1.56
G	(6, 6)	8.14	2.97
skupienie		punkt centralny	
▲		(0.0, 0.5)	
▲		(3.8, 4.0)	

Rysunek 6: Położenie centrów skupień na początku drugiej iteracji.

$$\bar{y} = \frac{y_A + y_B}{2} = \frac{0 + 1}{2} = 0.5$$

Nowy punkt centralny to (0.0, 0.5)

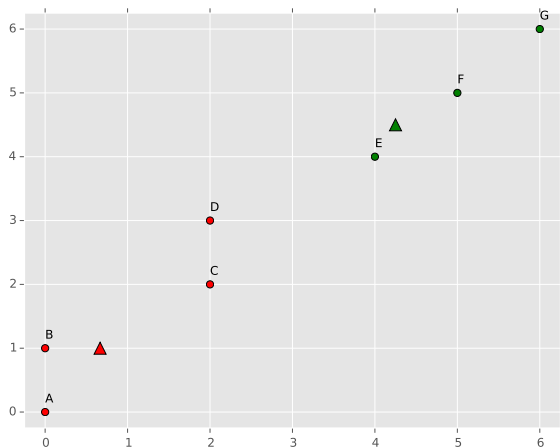
Skupienie 2 (▲): przydzielone są do niego punkty C, D, E, F i G.

$$\bar{x} = \frac{x_C + x_D + x_E + x_F + x_G}{5} = \frac{2 + 2 + 4 + 5 + 6}{5} = \frac{19}{5} = 3.8$$

$$\bar{y} = \frac{y_C + y_D + y_E + y_F + y_G}{5} = \frac{2 + 3 + 4 + 5 + 6}{5} = \frac{20}{5} = 4.0$$

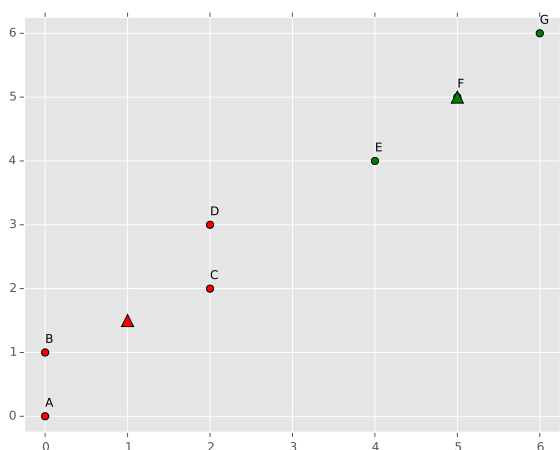
Nowy punkt centralny to (3.8, 4.0).

Na Rysunku 8 widzimy, że centra znalazły się w stabilnych pozycjach i już nie będą się dalej przesuwać. W tym momencie algorytm się kończy. Ostateczny wynik grupowania:



punkt		odległość ▲	odległość ▲
A	(0, 0)	1.2	6.19
B	(0, 1)	0.67	5.51
C	(2, 2)	1.67	3.36
D	(2, 3)	2.4	2.7
E	(4, 4)	4.48	0.56
F	(5, 5)	5.9	0.9
G	(6, 6)	7.31	2.3
skupienie		punkt centralny	
▲		(0.67, 1.00)	
▲		(4.25, 4.50)	

Rysunek 7: Położenie centrów skupień na początku trzeciej iteracji.



punkt		odległość ▲	odległość ▲
A	(0, 0)	1.8	7.07
B	(0, 1)	1.12	6.4
C	(2, 2)	1.12	4.24
D	(2, 3)	1.8	3.61
E	(4, 4)	3.91	1.41
F	(5, 5)	5.32	0.0
G	(6, 6)	6.73	1.41
skupienie		punkt centralny	
▲		(1.0, 1.5)	
▲		(5.0, 5.0)	

Rysunek 8: Położenie centrów skupień na początku czwartej iteracji. Stwierdzamy w niej, że uśrednienie punktów w skupieniach nie przesunie żadnych centrów i algorytm się kończy.

skupienie	przydzielone punkty
Skupienie 1 (▲)	A, B, C, D
Skupienie 2 (▲)	E, F, G

10.4 k-medoids

k-medoids jest taką odmianą *k-means*, w której zamiast średniej nowy punkt centralny powstaje przez wybranie z danego skupienia punktu, który ma najmniejszą sumę odległości do wszystkich pozostałych punktów w tym skupieniu. Jedną z własności *k-medoids* jest więc to, że punkt centralny zawsze jest tożsamy z jakimś przykładem ze zbioru uczącego.

11 Apriori

Najbardziej standardowym zadaniem uczenia nienadzorowanego jest grupowanie. Nie jest to jednak jedyne zadanie, które może być realizowane przez ten typ algorytmów uczenia maszynowego. W tej sekcji omówimy zadanie **odkrywania asocjacji** na przykładzie algorytmu **Apriori**.

W zadaniu odkrywania asocjacji naszym celem jest odkrycie interesujących zależności lub korelacji w danych. Przykładowo, chcielibyśmy wiedzieć, jak często obecność jakiegoś elementu A powiązana jest z obecnością jakiegoś elementu B (jak często ze sobą *współwystępują*). Znalezione **asocjacje** (powiązania) zapisuje się zazwyczaj w postaci tzw. **reguł asocjacyjnych**:

$$X \rightarrow Y \quad (1)$$

gdzie X i Y są rozłącznymi zbiorami elementów z jakiejś dziedziny. Reguły wyglądają podobnie jak implikacja i intuicyjnie możemy tak o nich myśleć – reguła mówi nam z pewną „wiarygodnością” (którą sformalizujemy już wkrótce), że jeżeli w transakcji znajduje się X , to znajdzie się w niej również Y . Z każdą regułą asocjacyjną powiązane są dwie miary określające jej statystyczną istotność („wiarygodność”):

- **wsparcie reguły** (ang. *support*) – procentowy udział obserwacji zawierających w sobie zarówno X jak i Y .
- **ufność reguły** (ang. *confidence*) – określa, ile procentowo spośród obserwacji zawierających w sobie X zawiera również Y .

Podczas przetwarzania bardzo dużych zbiorów danych zazwyczaj znajdziemy ogromną liczbę asocjacji. Miary te pozwalają nam odrzucić te asocjacje, które nie są szczególnie istotne. Przed uruchomieniem algorytmu musimy podać minimalne wartości dla tych miar.

Przykład 11.1 — Obliczanie miar istotności. Mamy następujący zbiór obserwacji:

t_1	a, b, d
t_2	a, b, c, e
t_3	a, b, c, f

$wsparcie(\{a,b\}) = 3/3 = 1$ (wszystkie obserwacje zawierają a i b)

$wsparcie(\{a,b,c\}) = 2/3$ (tylko ok. 66,7% obserwacji zawiera jednocześnie a , b i c)

$wsparcie(\{e\}) = 1/3$ (tylko ok. 33,3% obserwacji zawiera element e)

$ufnosc(a \rightarrow b) = 3/3 = 1$ (jeżeli w obserwacji jest a , to zawsze będzie też b)

$ufnosc(a \rightarrow c) = 2/3 = 0.67$

$ufnosc(c \rightarrow a) = 2/2 = 1$

$ufnosc(a \wedge b \rightarrow c) = 2/3 = 0.67$

$ufnosc(d \rightarrow a \wedge b \wedge c) = 0/1 = 0$ (w obserwacjach zawierających d nigdy nie występuje podzbiór a,b,c)

11.1 Zarys działania

Naiwnym rozwiązaniem problemu odkrywania asocjacji byłoby wygenerowanie wszystkich możliwych reguł, obliczenie ich wsparć i ufności, a następnie sprawdzenie, czy spełniają narzucone przez nas kryteria istotności (np. wsparcie $> 10\%$, ufność $> 70\%$). Złożoność tego podejścia jest jednak wykładnicza.

W algorytmie **Apriori** proces przeszukiwania podzielony jest na dwie fazy:

1. Wygenerowanie wszystkich *zbiorów częstych* ze *wsparciem* większym niż zadany przez nas próg.
2. Z wygenerowanych w poprzednim punkcie zbiorów tworzone są wszystkie możliwe reguły, a następnie wybierane tylko te, które mają *ufność* większą od zadanego przez nas progu.

11.2 Znajdowanie zbiorów częstych

Zbiory częste to zbiory elementów, które często ze sobą współwystępują. Pierwszą fazą w Apriori jest znalezienie wszystkich zbiorów częstych w danych. Następnie na ich podstawie wytworzone zostaną reguły asocjacyjne. Omówimy obie te fazy na przykładzie.

Określamy na początku, że minimalne wsparcie ma wynosić 0.3, a minimalna ufność 0.8. Zbiór transakcji (obserwacji) przedstawiony jest w poniższej tabelce. Litery A, \dots, D to oznaczenia produktów.

id	produkty
t_1	C, D
t_2	A, B, C
t_3	D
t_4	A, C, D
t_5	A, B, C

Zaczynamy od analizy pojedynczych produktów. Dla każdego z nich liczymy, w ilu transakcjach został zakupiony. Wyniki kolejnych kroków będziemy notować w tabelkach. Poniżej przedstawione są zakończone obliczenia dla jednoelementowych zbiorów produktów (będziemy konstruować coraz to większe zbiory).

produkty	trans. wspierające	wsparcie	
A	3	0.6	> 0.3 :)
B	2	0.4	> 0.3 :)
C	4	0.8	> 0.3 :)
D	3	0.6	> 0.3 :)

Liczba transakcji wspierających to liczba transakcji, w których występuje dana kombinacja produktów. Wsparcie to liczba transakcji wspierających podzielona przez łączną liczbę transakcji, czyli w tym wypadku przez 5. Wszystkie wartości wsparcia są powyżej progu, tak więc wszystkie zbiory produktów przechodzą do fazy tworzenia zbiorów dwuelementowych.

W następnym kroku tworzymy wszystkie dopuszczalne kombinacje zbiorów dwuelementowych złożonych z „zaakceptowanych” zbiorów jednoelementowych. Ponownie, w tabelce wypisane są zbiory produktów i wsparcia (spróbuj samemu policzyć je dla jakiegoś przypadku by sprawdzić, czy rozumiesz skąd się biorą).

produkty	trans. wspierające	wsparcie	
A, B	2	0.4	> 0.3 :)
A, C	3	0.6	> 0.3 :)
A, D	1	0.2	< 0.3 :(
B, C	2	0.4	> 0.3 :)
B, D	0	0.0	< 0.3 :(
C, D	2	0.4	> 0.3 :)

Jak widzimy, niektóre 2-elementowe kombinacje produktów występują niewystarczająco często w naszym zbiorze transakcji (mają wsparcie niższe od naszego założonego progu). Wykluczamy je z dalszej analizy. Wiemy również, że wszystkie zbiory większych rozmiarów zawierające jako podzbiór A, D lub B, D również będą poniżej progu wsparcia. Dlaczego? Ponieważ nie są w stanie zdobyć większej liczby transakcji wspierających.

Teraz kolej na zbiory trójelementowe. Trzeba najpierw wspomnieć o pewnej zasadzie, którą wcześniej przemilczeliśmy. Mianowicie, **wszystkie podzbiory rozpatrywanych zbiorów produktów muszą mieć wsparcie powyżej progu**. Innymi słowy, nie mogą być zaznaczone kolorem czerwonym w naszych tabelkach. Wychodzi nam z tego, że powstanie tylko 1 element trójelementowy. Pozostałe będą zawierać w sobie jakiś wykluczony „czerwony podzbiór” produktów (rozpisz te kombinacje i sprawdź, czy rzeczywiście tak będzie).

produkty	trans. wspierające	wsparcie
A, B, C	2	0.4

> 0.3 :))

Tutaj zakończy się działanie algorytmu. Jak to, zapytasz, a co ze zbiorem 4-elementowym? Otóż nie jesteśmy w stanie stworzyć tego zbioru w taki sposób, by nie zawierał żadnego wykluczonego przez brak wsparcia podzbioru. Widać od razu, że zbiór 4-elementowy A, B, C, D musiałby zawierać zarówno A, D jak i B, D . Nie mógłby mieć więc większego wsparcia niż któryś z nich. B, D ma wsparcie 0, tak więc bez rozpisywania wiemy, że zbiór A, B, C, D też będzie mieć wsparcie 0. Jest to na pewno prawda – w tabelce transakcji podanej na wejście algorytmu nie mamy żadnej transakcji 4-elementowej.

11.3 Tworzenie reguł asocjacyjnych

Drugą fazą algorytmu Apriori jest tworzenie reprezentacji „wiedzy”, czyli odpowiednich reguł asocjacyjnych. Rozpatrywane będą wszystkie niewykluczone zbiory z poprzednich tabel, za wyjątkiem zbiorów 1-elementowych, gdyż reguły dla nich nie miałyby żadnego poprzednika.

Zbiór częsty postaci $\{X, Y\}$ może doprowadzić do powstania dwóch reguł: $X \rightarrow Y$ i $Y \rightarrow X$. Zbiór częsty postaci $\{X, Y, Z\}$ może doprowadzić do powstania sześciu reguł:

- $X \rightarrow Y \wedge Z$,
- $Y \wedge Z \rightarrow X$,
- $Y \rightarrow X \wedge Z$,
- $X \wedge Z \rightarrow Y$,
- $Z \rightarrow X \wedge Y$,
- $X \wedge Y \rightarrow Z$.

Reguły dla większych zbiorów tworzone są analogicznie.

Dla każdej reguły wyliczana jest ufność i porównywana z minimalnym progiem. Reguły z ufnością poniżej progu zostają wykluczone i w tabelce są oznaczone czerwonym kolorem. Pozostałe reguły są „wiarygodne” i mówią nam o znalezionych istotnych zależnościach w danych. Poniższa tabelka zawiera policzone ufności wszystkich wygenerowanych reguł.

zbiór częsty	wsparcie	reguła	ufność
A, B	0.4	$A \rightarrow B$	$2/3 = 0.67 < 0.7 :($
A, B	0.4	$B \rightarrow A$	$2/2 = 1.00 > 0.7 :)$
A, C	0.6	$A \rightarrow C$	$3/3 = 1.00 > 0.7 :)$
A, C	0.6	$C \rightarrow A$	$3/4 = 0.75 > 0.7 :)$
B, C	0.4	$B \rightarrow C$	$2/2 = 1.00 > 0.7 :)$
B, C	0.4	$C \rightarrow B$	$2/4 = 0.50 < 0.7 :($
C, D	0.4	$C \rightarrow D$	$2/4 = 0.50 < 0.7 :($
C, D	0.4	$D \rightarrow C$	$2/3 = 0.67 < 0.7 :($
A, B, C	0.4	$A \rightarrow B \wedge C$	$2/3 = 0.67 < 0.7 :($
A, B, C	0.4	$B \wedge C \rightarrow A$	$2/2 = 1.00 > 0.7 :)$
A, B, C	0.4	$B \rightarrow A \wedge C$	$2/2 = 1.00 > 0.7 :)$
A, B, C	0.4	$A \wedge C \rightarrow B$	$2/3 = 0.67 < 0.7 :($
A, B, C	0.4	$C \rightarrow A \wedge B$	$2/4 = 0.50 < 0.7 :($
A, B, C	0.4	$A \wedge B \rightarrow C$	$2/2 = 1.00 > 0.7 :)$

Powyżej opisany został ogólny algorytm tworzenia reguł asocjacyjnych na podstawie zbiorów częstych. Można jednak go ulepszyć poprzez obserwację, że jeżeli pewne reguły nie spełniają kryterium minimalnej ufności, to automatycznie „pociągają” za sobą od razu pewne inne, bardziej złożone reguły (np. $A \rightarrow B$ pociągnęłyby za sobą $A \rightarrow B \wedge C$ i mielibyśmy mniej ufności do liczenia).

Podsumowując, Apriori nie jest trywialnym algorytmem, ale jeżeli uważnie prześledzisz procedurę generowania zbiorów częstych i reguł, oraz spróbujesz ją wykonać samodzielnie, to prawdopodobnie odkryjesz, że nie jest bardzo skomplikowany.

12 Drzewa decyzyjne

W tej sekcji omówione zostanie jedno z najpopularniejszych podejść **uczenia symbolicznego**, czyli uczenia, w którym wiedza zapisywana jest w postaci łatwo interpretowalnej przez człowieka.

Gdybyśmy sami wymyślali algorytmy uczenia maszynowego, to dość łatwo byśmy wymyślili drzewa decyzyjne. Bazowa koncepcja za nimi stojąca jest prosta i „naturalna”, często wykorzystywana w praktyce przy różnych okazjach przez ludzi, którzy nie rozpoznaliby uczenia maszynowego nawet gdyby ugryzło ich w stopę⁵. Rozumowalibyśmy mniej więcej tak:

1. W sumie byłoby fajnie, jakbyśmy mogli zadawać przypadkom „pytania” i na podstawie ich odpowiedzi powstawałyby coraz to bardziej zawężone podgrupy przypadków o takich samych odpowiedziach. Tak zazwyczaj klasyfikuje się ludzi na różne kategorie, takie jak np. 'dostanie wizę' i 'nie dostanie wizy', albo 'dostanie tę pracę' i 'nie dostanie tej pracy'.
2. Po zadaniu jakiegoś pytania podzielimy nasze przypadki na osobne podgrupy w zależności od ich odpowiedzi. Na przykład wszyscy, którzy stwierdzą, że są terrorystami, będą w jednej podgrupie, a pozostali w drugiej. Jak następnie zadamy pytanie, czy mogą dać dużo pieniędzy na łapówkę, to będą już cztery podgrupy. Wizy na pewno nie dostanie podgrupa terrorystów, których nie stać na łapówkę. W co niektórych krajach nie-terrorysty również nie dostaną wizy, jeżeli nie dadzą łapówki (wtedy pytanie o terroryzm jest nadmiarowe, gdyż nie ma wpływu na decyzję!). :)
3. Prędzej czy później w którejś podgrupie znajdą się tylko przypadki, które mają dokładnie tę samą interesującą nas decyzję, kategorię (np. każdy w podgrupie „łapówkarzy” siłą rzeczy należy do kategorii 'dostanie wizę'). Wtedy nie musimy zadawać im więcej pytań.
4. Pytania w naszym kontekście uczenia maszynowego oczywiście musiałyby dotyczyć wartości atrybutów, ponieważ atrybuty są jedynym źródłem informacji dostępnym dla algorytmów uczenia maszynowego.
5. Kolejna obserwacja: jak podzielimy przypadki na podgrupy po pierwszym pytaniu, to nie jesteśmy zmuszeni każdej grupie zadawać takiego samego następnego pytania. Możemy usprawnić proces i zadawać takie pytania, które **możliwie szybko doprowadzą do jasnego z punktu decyzji podziału**.

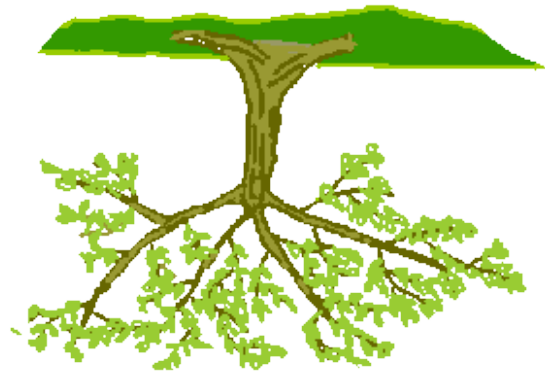
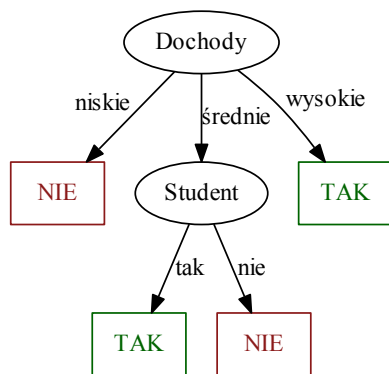
Takie mniej więcej rozumowanie (które pewnie zbyt niejasno przedstawiłem mimo jego prostoty) doprowadziłoby nas do koncepcji drzewa decyzyjnego, tylko nie nadalibyśmy naszemu podejściu takiej ładnej nazwy. Drzewo decyzyjne (jako graf) to wyłącznie pewna wizualizacja, perspektywa na myślenie o opisanym wyżej pomysle na dzielenie przypadków na coraz bardziej to uszczegółowione podgrupy.

12.1 Budowa drzewa

Na Rysunku 9 przedstawione jest przykładowe drzewo decyzyjne dla problemu określania, czy klient kupi jakiś towar. Z kolei na Rysunku 10 znajduje się „pomoc” w interpretacji grafu. Trzeba bowiem pamiętać, że drzewa decyzyjne to taki gatunek drzew, który rośnie do góry nogami. Oczywiście jest to kwestia umowy, czy korzeń w drzewie decyzyjnym będziemy rysować na dole (i wtedy kolejne węzły będą piąć się w górę) czy też na górze.

W ogólności drzewo jest *grafem* (bardzo ważne pojęcie w informatyce) i składa się z węzłów oraz krawędzi je łączących. Omówimy teraz budowę drzewa decyzyjnego na przykładzie Rysunku 9.

⁵Mamy nadzieję, że dzięki temu przedmiotowi będziesz potrafił/a rozpoznać uczenie maszynowe w tej i wielu innych sytuacjach. Może nawet się z nim zaprzyjaźnisz? :)



Rysunek 9: Przykładowe drzewo decyzyjne. Rysunek 10: Drzewa decyzyjne rosną ku dołowi.

węzeł – na rysunku elipsy lub prostokąty. W węźle znajduje się zawsze albo test na wartości pewnego atrybutu (na przykładzie elipsa), albo klasa decyzyjna (na przykładzie prostokąt). Oczywiście dokładny kształt nie ma znaczenia i różne figury geometryczne zostały wprowadzone by ułatwić rozróżnianie węzłów.

krawędź – łączy dwa węzły, przy czym strzałka określa kierunek połączenia. Z każdą krawędzią związana jest pewna **etykieta**, np. *niskie*, *średnie*, *tak*, *nie*. Etykiety to możliwe wartości atrybutu związanego z węzłem, z którego wyszła ta krawędź.

liść – taki węzeł, z którego nie wychodzi żadna krawędź. W liściu *zawsze* znajdować się będzie przypisanie do jakiejś klasy decyzyjnej, w tym wypadku do *TAK* albo do *NIE*. Na rysunku liście to węzły w kształcie prostokątów.

korzeń – od korzenia drzewo zaczyna rosnąć. Jest to węzeł, do którego nie dochodzi żadna krawędź, czyli w tym wypadku *Dochody*.

12.2 Klasyfikacja przykładów

Skupmy się na jakiejś ścieżce w drzewie, np. *Dochody* → *niskie* → *NIE*. Węzeł *Dochody* to test na wartość atrybutu *Dochody*. Nowe przypadki z wartością *niskie* na tym atrybucie przejdą w drzewie po krawędzi *niskie*, *średnie* po krawędzi *średnie* itd. Po przejściu krawędzią przypadek może napotkać na kolejny test, tym razem na innym atrybucie, i procedura się powtarza. Prędzej czy później każdy przypadek skończy w jakimś liściu zawierającym przydział do klasy decyzyjnej (w każdym kroku schodzimy w dół a drzewo ma skończoną wysokość). Przypadek jest *zawsze* przydzielany do klasy decyzyjnej zawartej w liściu, na którym zakończył „podróż”.

Drzewo decyzyjne można alternatywnie zapisać w postaci reguł określających przydział obiektów do klas. Każda ścieżka od korzenia do liścia odpowiada jednej regule.

Przykład 12.1 — **Zapisywanie drzewa w postaci reguł.** Przedstawimy drzewo z Rysunku 9 w postaci reguł.

jeżeli *Dochody* = *niskie* to *Decyzja* = *NIE*
 jeżeli *Dochody* = *wysokie* to *Decyzja* = *TAK*
 jeżeli *Dochody* = *średnie* ∧ *Student* = *tak* to *Decyzja* = *TAK*
 jeżeli *Dochody* = *średnie* ∧ *Student* = *nie* to *Decyzja* = *NIE*

Równoważnie, reguły można zapisywać w ten sposób:

$$\forall_x \text{Dochody}(x, \text{niskie}) \implies \text{Decyzja}(x, \text{NIE}),$$

gdzie dziedziną x jest zbiór wszystkich możliwych przypadków. ■

Reguły pozwalają zobaczyć, czym „tak naprawdę” jest drzewo decyzyjne oraz w jaki sposób dokonywana jest klasyfikacja obiektów. W drzewach decyzyjnych wiedza reprezentowana jest zasadniczo w postaci pewnych zdań logicznych. **Tego typu reprezentacja jest łatwa do interpretacji przez człowieka, co jest dużą zaletą tych metod.**

12.3 Entropia

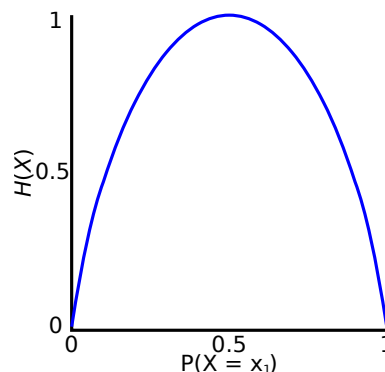
Zanim powiemy, jak tworzone są drzewa decyzyjne, musimy wyjaśnić pojęcie **entropii**. Pochodzi ono z teorii informacji i wykorzystane zostanie jako heurystyka do wyboru najlepszego w danym momencie atrybutu do podziału zbioru

Załóżmy, że mamy zmienną losową X o zbiorze wartości $\{x_1, x_2, \dots, x_n\}$. Intuicyjnie, zmienna losowa to funkcja, która zwraca którąś ze swoich dopuszczalnych wartości (zgodnie z pewnym rozkładem prawdopodobieństwa). Entropię tej zmiennej wyliczymy ze wzoru⁶:

$$H(X) = \sum_{i=1}^n p(x_i) \cdot \log_r \frac{1}{p(x_i)} = - \sum_{i=1}^n p(x_i) \cdot \log_r p(x_i)$$

Podstawa logarytmu r wyraża nam liczbę bazowych stanów, za pomocą których możemy kodować informację. Zazwyczaj przyjmuje się $r = 2$ (jak pewnie wiesz, w komputerach wszystko zapisywane jest w kodzie binarnym, czyli za pomocą samych 0 i 1). $p(x_i)$ to z kolei prawdopodobieństwo wylosowania wartości x_i .

Wykres entropii w przypadku **dwóch** możliwych wartości (x_1, x_2) zmiennej losowej X , dla różnych prawdopodobieństw wylosowania x_1 na osi x wykresu (z założeń wiemy, że $p(x_2) = 1 - p(x_1)$), wygląda następująco:



Jak można zauważyć, entropia osiąga 0 w przypadku, gdy któraś z wartości zmiennej losowej X ma prawdopodobieństwo wylosowania równe 1. Jeżeli wartości są równo prawdopodobne ($p(x_1) = 0.5, p(x_2) = 0.5$), to entropia osiąga maksimum, czyli wartość 1.

Przykład 12.2 — Obliczanie entropii. Załóżmy, że zmienna losowa X przyjmuje:

- wartość x_1 z prawdopodobieństwem $\frac{2}{3}$ i x_2 z prawdopodobieństwem $\frac{1}{3}$:

$$H(X) = -\frac{2}{3} \cdot \log_2 \frac{2}{3} - \frac{1}{3} \cdot \log_2 \frac{1}{3} \approx -\frac{2}{3} \cdot (-0.5849) - \frac{1}{3} \cdot (-1.5849) \approx 0.918$$

- wartość x_1 z prawdop. 1 i x_2 z prawdop. 0:

$$H(X) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0 - 0 = 0$$

- wartość x_1 z prawdop. 0.5 i x_2 z prawdop. 0.5:

$$H(X) = -0.5 \cdot \log_2 0.5 - 0.5 \cdot \log_2 0.5 = -0.5 \cdot (-1) - 0.5 \cdot (-1) = 0.5 + 0.5 = 1$$

⁶Literka H wzięła się z inspiracji H -theorem wprowadzoną przez Boltzmana (dziedzina termodynamiki).

Warto jeszcze coś wspomnieć o interpretacji entropii. Można ją traktować jako miarę nieprzewidywalności, „nieuporządkowania” zmiennej losowej. W drzewach decyzyjnych chcemy ją minimalizować.

12.4 Uczenie – Algorytm ID3

Informacje wstępne

Celem uczenia drzew decyzyjnych jest utworzenie **jak najmniejszego** drzewa. Łatwo zauważyć, że stworzenie na podstawie przykładów uczących *jakiegoś* drzewa jest bardzo proste – wystarczy utworzyć osobną ścieżkę dla każdego przykładu uczącego. Jednak takie drzewa są obciążające obliczeniowo podczas klasyfikacji i mają słabą zdolność uogólniania (powstały model nie dostarcza nam żadnej interesującej wiedzy). Dlatego jesteśmy zainteresowani tym, by opisać nasze przykłady uczące możliwie niewielkim drzewem (można tu zauważyć pewną analogię do brzytwy Ockhama).

W celu utworzenia minimalnego drzewa można by zastosować algorytm dokładny, jednak złożoność takiego algorytmu byłaby wykładnicza. W praktyce nie musimy posiadać absolutnie najmniejszego drzewa i zadowolamy się „dość małym” drzewem wygenerowanym przez heurystykę. Heurystyka ta oparta jest właśnie na mierze entropii.

Najbardziej podstawowym algorytmem uczenia drzew decyzyjnych jest ID3 (*Iterative Dichotomiser 3*). Dychotomiczny podział zbioru to taki jego podział na 2 podzbiory A i B, że nie mają one części wspólnej i po zsumowaniu dadzą zbiór wyjściowy.

Omówienie na przykładzie

Będziemy rozważać zbiór danych uczących S przedstawiony w poniższej tabelce. Interesujące nas atrybuty *Matematyka*, *Biologia* i *Polski* przyjmują wartości ze zbioru $\{3, 4, 5\}$. *Uczeń* jest identyfikatorem (nie bierze udziału w uczeniu), a *Decyzja* określa prawdziwe klasy decyzyjne przykładów. Problemem jest odpowiedzenie na pytanie, czy uczeń dostanie stypendium biorąc pod uwagę jego oceny.

Uczeń	Matematyka	Biologia	Polski	Decyzja
A	4	4	5	TAK
B	4	5	4	TAK
C	3	4	4	NIE
D	5	3	5	NIE
E	4	4	4	NIE
F	3	5	3	NIE

Czy potrafisz odgadnąć, jaką metodą ktoś podejmował te decyzje? Odpowiedź: stypendia przyznawane były tym studentom, którzy mieli z przynajmniej jednego przedmiotu 5 i nie dostali żadnej oceny niższej niż 4.

Entropia ze względu na decyzje tego całego zbioru ($\{T, T, N, N, N, N\}$) wynosi:

$$H(S) = -\frac{2}{6} \cdot \log_2 \frac{2}{6} - \frac{4}{6} \cdot \log_2 \frac{4}{6} \approx 0.917$$



Konstrukcję drzewa zawsze zaczynamy od utworzenia korzenia. Sprawdzamy, czy wszystkie przykłady A,B,C,D,E,F są zaklasyfikowane do tej samej klasy decyzyjnej. W tym wypadku nie są, więc szukamy atrybutu najlepiej dzielącego przykłady ze względu na miarę entropii. Obliczenia zaczniemy od *Matematyki*.

Wypisujemy podziały przypadków ze względu na ich wartości na atrybucie *Matematyka*:

wartość atrybutu	częstość	przypadki	entropia
3	2/6	C(NIE), F(NIE)	0
4	3/6	A(TAK), B(TAK), E(NIE)	0.917
5	1/6	D(NIE)	0

Entropie zostały policzone ze względu na rozkład decyzji w przypadkach. Dla wartości 4 entropia przypadkowo wyszła taka sama jak $H(S)$, ale nie ma to szczególnego znaczenia. W ogólności entropia w poszczególnych gałęziach może wyjść dowolna (nawet większa), jednak jak uśrednimy wszystko ważąc po częstościach to i tak $H(S|Matematyka)$ nie przekroczy $H(S)$. Entropia ze względu na podział (który oznaczamy poniżej symbolem '|') na atrybucie *Matematyka* będzie średnią arytmetyczną wszystkich entropii z tabelki ważoną po odpowiadających im częstościach:

$$H(S|Matematyka) = 2/6 \cdot 0 + 3/6 \cdot 0.917 + 1/6 \cdot 0 \approx 0.459$$

Analogiczne tabelki i wyliczenia możemy wykonać dla atrybutów *Biologia* i *Polski*. Możesz je zrobić jako ćwiczenie, entropie powinny wyjść następujące:

$$H(S|Biologia) = 0.791$$

$$H(S|Polski) = 0.791$$

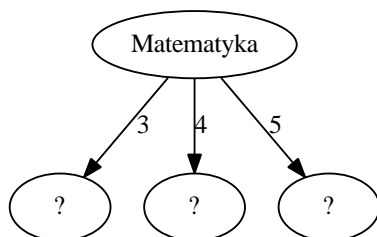
W algorytmie ID3 wykorzystuje się miarę **InformationGain** (zapisywane we wzorach jako *InfoGain*). Oblicza się ją poprzez odjęcie od początkowej entropii ($H(S)$) entropii po podziale na pewnym atrybucie (np. $H(S|Biologia)$). *InformationGain* wyraża, ile informacji „zyskaliśmy” na pojedynczym podziale.

$$InfoGain(S|Matematyka) = H(S) - H(S|Matematyka) = 0.917 - 0.459 = 0.458$$

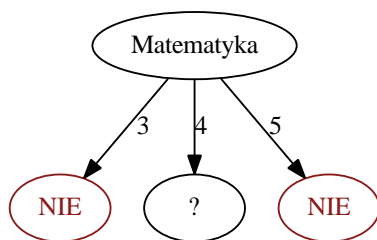
$$InfoGain(S|Biologia) = H(S) - H(S|Biologia) = 0.917 - 0.791 = 0.126$$

$$InfoGain(S|Polski) = H(S) - H(S|Polski) = 0.917 - 0.791 = 0.126$$

Jak widzimy, zysk największy był w podziale na *Matematykę*, więc ten atrybut umieszczamy w korzeniu i rysujemy odpowiednie krawędzie pamiętając o etykietach.



Umieszczamy w korzeniu atrybut *Matematyka* i tworzymy gałęzie dla wszystkich przyjmowanych przez niego wartości (3, 4, 5).



Sprawdzamy, czy w danym podziale ze względu na ocenę z *Matematyki* elementy należą do tej samej klasy decyzyjnej. Tak jest dla zbiorów zawierających przykłady z ocenami 3 i 5, więc możemy utworzyć liście z decyzjami. W zbiorze przykładów z oceną 4 są dwa przypadki TAK i jeden NIE, tak więc będziemy musieli rozpatrywać podział dla zbioru przypadków $\{A, B, E\}$ ze względu na pozostałe atrybuty: *Biologia*, *Polski*.

Rozpatrujemy więc podziały na zbiorze $S = \{A, B, E\}$ (tylko one miały 4 z *Matematyki*). Entropię ($H(S)$) w tym zbiorze możemy odczytać w pierwszej tabelce i jest to 0.917. Możemy zrobić kolejne tabelki podziałów:

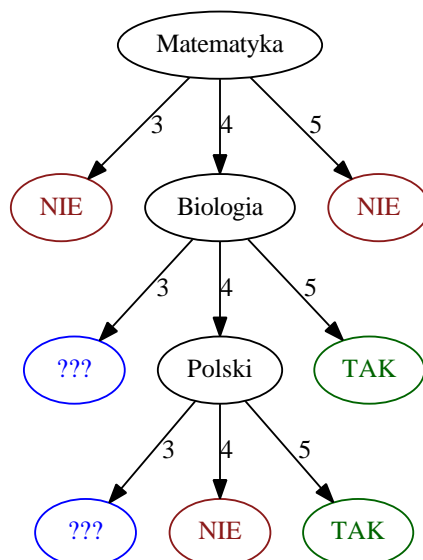
	wartość atrybutu	częstość	przypadki	entropia
Biologia:	3	0/3		
	4	2/3	A(TAK), E(NIE)	1
	5	1/3	B(TAK)	0
	wartość atrybutu	częstość	przypadki	entropia
Polski:	3	0/3		
	4	2/3	B(TAK), E(NIE)	1
	5	1/3	A(TAK)	0

$$InfoGain(S|Biologia) = H(S) - H(S|Biologia) = 0.917 - 2/3 \cdot 1 \approx 0.25$$

$$InfoGain(S|Polski) = H(S) - H(S|Polski) = 0.917 - 2/3 \cdot 1 \approx 0.25$$

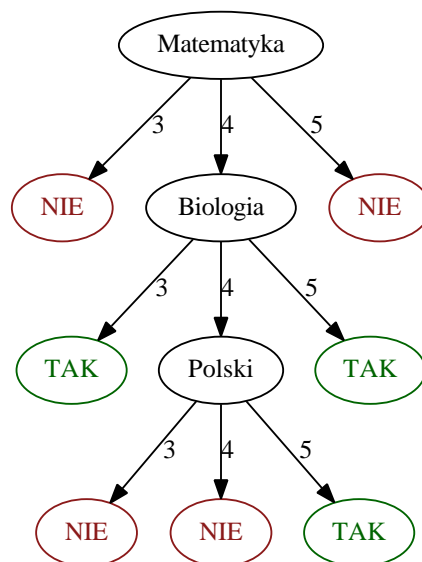
Nie ma znaczenia, który atrybut wybierzemy do podziału (wartości *InfoGain* są równe) – wybieramy więc, zgodnie z kolejnością, *Biologię*. Zbiór dla oceny 4 z *Biologii* nadal nie zawiera przykładów z tylko jednej klasy decyzyjnej, więc wykonujemy dla niego jeszcze podział dla *Polskiego* (ostatni atrybut, jaki nam został).

Prawie-ostateczne drzewo wygląda tak:



Węzły **???** nie zawierają żadnych przykładów, więc nie możemy jednoznacznie stwierdzić, którą klasę decyzyjną umieścić w takim liściu. Domyślnie bierze się klasę najczęściej występującą na danym „poziomie”. Dla 4 z *Matematyki* najczęściej występuje **TAK**, a dla 4 z *Matematyki* i 4 z *Biologii* **NIE** występuje tak samo często jak **TAK**. W tym drugim przypadku arbitralnie weźmiemy **NIE**.

Ostateczne drzewo (klasyfikator) przedstawione jest poniżej.



Komentarz

- Na żadnym etapie uczenia nie było brane pod uwagę uporządkowanie ocen. Wynika z tego, że **algorytm ID3 traktuje wszystkie dane tak jakby były nominalne**.
- Paradoksalnie, gdyby decyzje o stypendium były podejmowane przez nasz wyprodukowany klasyfikator, to ocena 5 z *Matematyki* automatycznie przekreślałaby na nie szanse! :) Z kolei osoba mająca 4 z *Matematyki* i 3 z *Biologii* dostałaby stypendium niezależnie od oceny z polskiego. **Wynika to przede wszystkim z wybrakowanych danych uczących**. W uczeniu maszynowym algorytm nauczy się tylko tego, co mu pokażemy, tak więc odpowiednio liczny i reprezentatywny zbiór uczący to podstawa. W naszym zbiorze uczącym 5 z *Matematyki* miała tylko jedna osoba, jednak nie dostała ona stypendium przez słabą ocenę z innego przedmiotu. Decyzja algorytmu w kontekście danych, z którymi pracował, była słuszna.

12.5 Uczenie – Algorytm C4.5

Algorytm **C4.5** jest ulepszoną wersją ID3. Ulepszenia obejmują:

- wprowadzenie miary *ilorazu przyrostu informacji* zamiast zysku informacji (*InfoGain*) w celu „karania” atrybutów o wielu różnych wartościach (są one niejawnie preferowane w ID3),
- wbudowane radzenie sobie z brakującymi wartościami (nie są brane pod uwagę podczas liczenia miar entropii),
- obsługa ciągłych wartości (podział na przedziały),
- upraszczanie drzewa po jego utworzeniu (post-pruning). Polega ono na usuwaniu któregoś z węzłów i wstawianiu na jego miejsce liścia zawierającego decyzję. Jeżeli taka zmiana poprawia jakość klasyfikacji na zbiorze testowym, to zmiana jest zachowywana.

Iloraz przyrostu informacji

Zanim zdefiniujemy **iloraz przyrostu informacji**, musimy wprowadzić **podział informacji** (*Split*). *Podział informacji* jest po prostu obliczeniem entropii dla zbioru wartości pewnego atrybutu (poprzednio liczyliśmy entropię ze względu na klasy decyzyjne).

Przykład 12.3 — Obliczanie podziału informacji. Wykorzystamy niedawno rozważaną tabelę opisującą uczniów i decyzje o stypendiach. Dla atrybutu *Matematyka* zbiór (właściwie to multizbiór) wartości wygląda następująco: {5, 4, 4, 4, 3, 3}. Entropię tego zbioru, czyli podział informacji, możemy policzyć jak każdą inną:

$$Split(S|Matematyka) = -1/6 \cdot \log_2 1/6 - 3/6 \cdot \log_2 3/6 - 2/6 \cdot \log_2 2/6 \approx 1.46$$

Zauważ, że entropia wyszła większa niż 1. Jest to normalne w przypadku sytuacji, w których mamy więcej niż dwie różne wartości w zbiorze (tu mieliśmy trzy: 5, 4 i 3). ■

Iloraz przyrostu informacji (*GainRatio*) wyraża się wzorem:

$$GainRatio(S|Atrybut) = \frac{InfoGain(S|Atrybut)}{Split(S|Atrybut)}$$

Podobnie jak przy zysku informacji chcemy go maksymalizować. Dzielenie przez *podział informacji*, czyli *Split*, sprawia, że wyrównane będą szanse atrybutów z dużą i małą liczbą różnych wartości na zbiorze przykładów uczących (ID3 niejawnie premiowało atrybuty o wielu przyjmowanych wartościach).

13 Naiwny klasyfikator Bayesa

Naiwny klasyfikator Bayesa to klasyfikator opierający swoje przewidywania na rachunku prawdopodobieństwa. W związku z tym pojawi się parę wzorów matematycznych, ale się nie martw – wszystko, co będzie potrzebne, będzie wcześniej przypomniane, a same wzory gdy się już je zrozumie są naprawdę proste!

Wbrew swojej nazwie, w niektórych zastosowaniach (takich jak np. klasyfikacja tekstów) naiwny klasyfikator bayesowski daje sobie radę bardzo dobrze! Jego „naiwność” polega na założeniu o niezależności atrybutów. Innymi słowy, będziemy zakładać, że znając wartość jakichś atrybutów, nie jesteśmy w stanie nic powiedzieć o wartościach innych atrybutów, tzn. nie istnieje między ich wartościami żaden związek. Przykład sytuacji w której takie założenie byłoby prawdziwe to np. wyniki kolejnych rzutów kostką: nie istnieje żaden związek pomiędzy wartościami, które wyrzuciliśmy za pierwszym i za drugim razem. W praktyce jednak atrybuty będą niemal zawsze ze sobą w pewien sposób powiązane, np. jeśli wiemy, że temperatura powietrza jest wysoka, to jest wysokie prawdopodobieństwo, że na dworze jest słonecznie.

13.1 Prawdopodobieństwo, prawdopodobieństwo warunkowe

Aby się upewnić, że przy próbie zrozumienia naiwnego klasyfikatora Bayesa nie napotkamy na żadne większe problemy, przypomnijmy sobie najpierw kilka pojęć z matematyki:

Prawdopodobieństwo zdarzenia możemy rozumieć jako procent prób, w których dane zdarzenie zajdzie.

Pomyśl o tym w ten sposób: jeśli mamy skrzynkę z kolorowymi piłkami (trzy niebieskie, siedem czerwonych), to prawdopodobieństwo wylosowania niebieskiej piłki wyniesie $\frac{3}{3+7} = 30\%$, tzn. wyciągniemy ze skrzynki niebieską piłkę w średnio trzech na dziesięć przypadków. Podobnie możemy liczyć prawdopodobieństwo wartości atrybutów na podstawie danych uczących: jeśli dla 10 przypadków uczących trzy z nich przyjmują dla pewnego atrybutu wartość A a siedem z nich przyjmuje wartość B , to możemy oczekiwać, iż nowe przykłady będą przyjmować wartość A również w 30% przypadków. Prawdopodobieństwo zdarzenia A będziemy oznaczać jako $P(A)$.

Prawdopodobieństwo warunkowe zdarzenia X pod warunkiem wystąpienia zdarzenia Y będziemy oznaczali jako $P(X|Y)$. Można je rozumieć podobnie do zwykłego prawdopodobieństwa, tym razem jednak interesuje nas procent liczony jedynie z takich prób, gdzie zaszło zdarzenie Y .

Odpowiedni przykład pomoże nam to dobrze zobrazować. Prawdopodobieństwo, że pewien losowy dzień będzie miał rano przymrozki wynosi dla Polski: $P(\text{przymrozki}) = \frac{100}{365} \approx 27\%$. Jeżeli jednak wiemy, że jest to dzień kwietniowy, możemy policzyć prawdopodobieństwo warunkowe: $P(\text{przymrozki}|\text{kwiecień}) = \frac{P(\text{przymrozki} \wedge \text{kwiecień})}{P(\text{kwiecień})} = \frac{5}{30} \approx 17\%$. Jak widać, różni się ono od zwykłego prawdopodobieństwa, ponieważ wiemy już coś więcej na temat naszego dnia – znamy miesiąc!

Podobnie można liczyć prawdopodobieństwa warunkowe dla danych uczących. Możemy na przykład policzyć prawdopodobieństwo tego, że warto grać w tenisa pod warunkiem, że jest ciepło: $P(\text{PlayTennis} = \text{YES}|\text{Temperature} = \text{hot})$. Patrzymy wtedy jedynie na przykłady gdzie $\text{Temperature} = \text{hot}$ i liczymy prawdopodobieństwo tego, że warto grać w tenisa, jedynie na ich podstawie.

13.2 Twierdzenie Bayesa

Twierdzenie Bayesa brzmi następująco:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}$$

Sposób wyprowadzenia go jest prosty. Jak zauważyliśmy przed chwilą, $P(Y|X) = \frac{P(Y \wedge X)}{P(X)}$. Podstawiając więc powyższe do prawej strony równania, otrzymamy:

$$P(X|Y) = \frac{P(Y \wedge X) \cdot P(X)}{P(Y) \cdot P(X)} = \frac{P(Y \wedge X)}{P(Y)}$$

Porównując wzory przed i po podstawieniu, możemy zauważyć, iż zmienił się tylko licznik wyrażenia, a więc $P(Y|X) \cdot P(X) = P(Y \wedge X)$. Analogicznie, możemy powiedzieć, że $P(X|Y) \cdot P(Y) = P(X \wedge Y)$. Jednocześnie, możesz zauważyć, iż $P(X \wedge Y) = P(Y \wedge X)$, jako że kolejność czynników w iloczynie logicznym nie ma znaczenia, a więc:

$$P(X|Y) = \frac{P(Y \wedge X)}{P(Y)} = \frac{P(X \wedge Y)}{P(Y)} = \frac{P(X|Y) \cdot P(Y)}{P(Y)} = P(X|Y)$$

Jak widzisz, wzory wydają się nie kłamać, a więc bezpiecznie będzie im zaufać. :)

13.3 Naiwny klasyfikator Bayesa

W jaki sposób będziemy decydować, którą klasę należy przypisać do nowego przykładu? Będziemy wybierać klasę najbardziej prawdopodobną. Ale jak obliczyć prawdopodobieństwo przynależności przykładu do klasy? Jak się możesz spodziewać, z wykorzystaniem twierdzenia Bayesa.

Prawdopodobieństwo tego, że przykład q jest klasy K^q możemy opisać jako prawdopodobieństwo klasy pod warunkiem wartości atrybutów, czyli:

$$P(K^q|A^q) = P(K^q|A_1^q \wedge A_2^q \wedge \dots \wedge A_n^q),$$

gdzie jako A^q rozumiemy wartości atrybutów przykładu q , np. A_1^q to wartość pierwszego atrybutu przykładu q . Niestety, tego typu prawdopodobieństwa nie są nam znane i trudno byłoby je obliczyć na podstawie danych uczących...

Użyjmy twierdzenia Bayesa!

$$P(K^q|A^q) = \frac{P(A^q|K^q) \cdot P(K^q)}{P(A^q)}$$

Prawdopodobieństwo $P(K^q)$ jest proste do wyliczenia – wystarczy stwierdzić, ile procent przykładów uczących posiada klasę K^q . Prawdopodobieństwo $P(A^q)$ byłoby już bardziej problematyczne, ale tak naprawdę nie musimy go znać! Zauważ, iż będzie ono wynosiło tyle samo bez względu na klasę, a więc ignorując je, nadal możemy porównywać między sobą to, która z klas jest bardziej prawdopodobna! Aby zachować formalną poprawność założymy, że $\frac{1}{P(A^q)} = C$. Wtedy:

$$P(K^q|A^q) = P(A^q|K^q) \cdot P(K^q) \cdot C,$$

gdzie wartość C nas nie interesuje.

Niestety, nadal nie jest nam znana wartość $P(A^q|K^q)$. W tym miejscu możemy użyć **naiwnego** założenia o niezależności atrybutów. Rachunek prawdopodobieństwa głosi, iż dla niezależnych atrybutów $A_1^q, A_2^q, \dots, A_n^q$ zachodzi:

$$P(A^q|K^q) = P(A_1^q \wedge A_2^q \wedge \dots \wedge A_n^q|K^q) = P(A_1^q|K^q) \cdot P(A_2^q|K^q) \cdot \dots \cdot P(A_n^q|K^q),$$

a więc:

$$P(K^q|A^q) = P(A_1^q|K^q) \cdot P(A_2^q|K^q) \cdot \dots \cdot P(A_n^q|K^q) \cdot P(K^q) \cdot C.$$

Jak wspominaliśmy wcześniej, w praktyce nasze atrybuty nie będą niezależne... ale dlatego właśnie mówimy o **naiwnym** Bayesie – ponieważ używamy bardzo **naiwnego** założenia!

Prawdopodobieństwa postaci $P(A_i^q|K^q)$ będą w praktyce znacznie prostsze do obliczenia niż $P(A^q|K^q)$, ponieważ potrzebujemy do nich mniej danych: częściej znajdziemy przykłady uczące które są zgodne na przynajmniej dwóch atrybutach z naszym przykładem testowym niż na wszystkich naraz!

A więc, co należy zrobić aby zaklasyfikować nowy przykład algorytmem naiwnego Bayesa?

1. Oblicz prawdopodobieństwa klas decyzyjnych $P(K^q)$.

2. Oblicz prawdopodobieństwa warunkowe wartości atrybutów pod warunkiem klas decyzyjnych $P(A_j^q|K^q)$.
3. Podstaw obliczone wyżej prawdopodobieństwa do wzorów na prawdopodobieństwo różnych klas pod warunkiem naszego przykładu testowego ($P(K^q|A^q)$).
4. Wybierz tę klasę, która osiąga najwyższą wartość prawdopodobieństwa! Pamiętaj, że wartość C nie będzie miała znaczenia – liczy się jedynie stojący przy niej mnożnik!

Jeżeli niektóre elementy naiwnego Bayesa nadal pozostają dla Ciebie niezrozumiałe, rzuć okiem na poniższy przykład. Mamy nadzieję, iż zobaczenie naiwnego Bayesa w akcji rozwieje wszelkie niepewności!

Przykład 13.1 — Naiwny Bayes w praktyce. Użyjmy znanych danych:

Day	Outlook	Temperature	Humidity	PlayTennis
D1	sunny	hot	high	NO
D2	sunny	hot	high	NO
D3	overcast	hot	high	YES
D4	overcast	hot	normal	YES
D5	sunny	hot	low	YES
D6	rain	mild	high	NO
D7	rain	mild	normal	NO

Załóżmy, że chcemy sklasyfikować nowy przykład:

Day	Outlook	Temperature	Humidity	PlayTennis
D8	sunny	hot	normal	?

Musimy więc obliczyć i porównać ze sobą prawdopodobieństwa dwóch różnych klas pod warunkiem tego przykładu:

$$P(YES|A^{D8}) = P(sunny|YES) \cdot P(hot|YES) \cdot P(normal|YES) \cdot P(YES) \cdot C,$$

$$P(NO|A^{D8}) = P(sunny|NO) \cdot P(hot|NO) \cdot P(normal|NO) \cdot P(NO) \cdot C.$$

Mamy więc do obliczenia osiem prawdopodobieństw:

- $P(YES) = 3/7$
- $P(NO) = 4/7$
- $P(sunny|YES) = 1/3$
- $P(hot|YES) = 1$
- $P(normal|YES) = 1/3$
- $P(sunny|NO) = 1/2$
- $P(hot|NO) = 1/2$
- $P(normal|NO) = 1/4$

Podstawiając do wzorów, otrzymujemy:

$$P(YES|A^{D8}) = \frac{1}{3} \cdot 1 \cdot \frac{1}{3} \cdot \frac{3}{7} \cdot C = \frac{1}{21} \cdot C$$

$$P(NO|A^{D8}) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{4}{7} \cdot C = \frac{1}{28} \cdot C$$

Prawdopodobieństwo jest wyższe dla klasy *YES*, a więc naiwny klasyfikator Bayesa powie nam, iż owszem, należy pójść zagrać w tenisa. ■

13.4 O zerowych prawdopodobieństwach warunkowych

Pozostaje jeszcze jedna kwestia związana z naiwnym Bayesem: obliczone przez nas prawdopodobieństwa warunkowe mogą czasem wynieść 0! Zauważ, że ostateczne prawdopodobieństwo uzyskujemy przez wymnażanie przez siebie wielu prawdopodobieństw warunkowych, a więc wystarczy aby jedno z nich wyniosło 0, by całe prawdopodobieństwo się również wyzerowało! W szczególności, jeśli wyzerują nam się prawdopodobieństwa dla wszystkich klas, nie możemy ich ze sobą porównać... bo są sobie równe (i wynoszą zero)! Aby uniknąć związanych z tym problemów, gdy któreś ze składowych prawdopodobieństw warunkowych wynosi zero, zmieniamy jego wartość na inną, niewielką. Na jaką konkretnie?

Jedną z najpopularniejszych metod jest tzw. **wygładzanie Laplace'a**. Przyjmuje ono jeden parametr k . Wygładzanie Laplace'a polega na zmodyfikowaniu prawdopodobieństw, tak jak gdyby zawsze znalazło się przynajmniej k reprezentantów każdej z wartości atrybutów. Im wyższa wartość k , tym mniejsze zróżnicowanie pomiędzy prawdopodobieństwami różnych zdarzeń: w naszym przypadku będziemy chcieli mieć je w miarę niskie, ponieważ nie chcemy tracić informacji zawartej w danych uczących, a jedynie przeciwdziałać zerowym prawdopodobieństwom.

Wcześniej, prawdopodobieństwa liczyliśmy jak poniżej:

$$P(K^q) = \frac{\text{liczbaPrzykladow}(K = K^q)}{\text{liczbaPrzykladow}()}$$

$$P(A_i^q|K^q) = \frac{\text{liczbaPrzykladow}(A_i = A_i^q \wedge K = K^q)}{\text{liczbaPrzykladow}(K = K^q)}$$

Po uwzględnieniu wygładzania Laplace'a, wzory się zmieniają:

$$P(K^q) = \frac{\text{liczbaPrzykladow}(K = K^q) + k}{\text{liczbaPrzykladow}() + k \cdot \text{liczbaWartosci}(K)}$$

$$P(A_i^q|K^q) = \frac{\text{liczbaPrzykladow}(A_i = A_i^q \wedge K = K^q) + k}{\text{liczbaPrzykladow}(K = K^q) + k \cdot \text{liczbaWartosci}(A_i)}$$

Wyobraź sobie zbiór uczący z trzema klasami, $\{K_1, K_2, K_3\}$, gdzie dane uczące zawierają 4 wystąpienia klasy K_1 , jedno wystąpienie klasy K_2 i zero wystąpień klasy K_3 . Bez wygładzania Laplace'a, prawdopodobieństwa tych klas wyglądałyby następująco:

$$P(K_1) = \frac{4}{5} = 80\%; P(K_2) = \frac{1}{5} = 20\%; P(K_3) = \frac{0}{5} = 0\%$$

Z wygładzaniem (dla $k = 1$), otrzymamy:

$$P(K_1) = \frac{4+1}{5+3} = 62.5\%; P(K_2) = \frac{1+1}{5+3} = 25\%; P(K_3) = \frac{0+1}{5+3} = 12.5\%$$

Jako przyczynę zmiany wzorów, możesz sobie wyobrazić dodanie do zbioru uczącego dodatkowych przykładów uczących, po k dla każdej wartości atrybutu. Dla powyższego przykładu, do danych uczących „dodaliśmy” po $k = 1$ przykładzie każdej klasy, a więc w nowych danych mamy $\text{liczbaPrzykladow}() + k \cdot \text{liczbaWartosci}(K) = 5 + 3 = 8$ przykładów.

14 Sztuczne sieci neuronowe

Bez wątplenia najbardziej niezwykłym znanym nam układem obliczeniowym jest mózg ludzki – około 1.5 kg wyspecjalizowanych tkanek zdolnych jest do przeprowadzania abstrakcyjnych rozumowań, przetwarzania języka naturalnego, przetwarzania obrazu, nadzoru nad ogromną liczbą procesów życiowych oraz wieloma innymi rzeczami, na których wymienienie nie starczyłoby miejsca. Dużo z wyżej wymienionych czynności, np. efektywne przetwarzanie języka naturalnego, wciąż stanowi problem dla nawet najlepszych algorytmów uczenia maszynowego, a jednak każdemu z nas przychodzi ono naturalnie. Wszystko wskazuje również na to, że mózg jest zdolny do tworzenia świadomości. Nie dziwi tedy, że w dziedzinie sztucznej inteligencji zainteresowanie mózgiem zawsze było bardzo duże. Niestety, jak wiemy zrozumienie działania mózgu jest wysoce nietrywialnym zadaniem.

Zamiast analizować mózg w celu odkrycia mechanizmów jego działania, można zastosować podejście syntetyczne, to znaczy można spróbować zbudować coś podobnego do mózgu z prostszych bazowych części. Poprzez sprawdzanie, które części są użyteczne, a które nie, oraz badanie ich wzajemnych interakcji, można się wiele dowiedzieć o działaniu samego skomplikowanego oryginalnego systemu. Dzięki neuroanatomii wiemy, że mózg realizuje swoje czynności dzięki interakcjom zasadniczo funkcyjnie prostych komórek, neuronów. Koncepcja sztucznego neuronu sięga mniej więcej lat 40. XX wieku, kiedy to zaczęto rozważać uproszczone wersje biologicznych neuronów w celu badania ich właściwości. Z takich uproszczonych neuronów można konstruować całe sieci neuronowe. Sieci te, po zastosowaniu algorytmu uczenia, można nauczyć wielu różnych rzeczy, w tym na przykład dokonywania klasyfikacji pewnych elementów.

Aktualnie sztuczne sieci neuronowe w odsłonie *deep learningu*⁷ doprowadziły do wzmożonego zainteresowania tą techniką uczenia maszynowego i osiągnięto dzięki nim wiele ciekawych rezultatów. Na szczególną uwagę zasługuje zwycięstwo 4:1 komputera *AlphaGo*, działającego w oparciu o głębokie sieci neuronowe, w meczu przeciwko Lee Sedol'owi, jednemu z najlepszych graczy Go na świecie. Dla niezorientowanych warto wspomnieć, że dla komputerów gra w Go jest o wiele trudniejsza niż gra w szachy, i wielu wieszczyło, że jeszcze długo na tym polu ludzie będą wiedli prym nad maszynami. . .

14.1 Sztuczny neuron

Jak już zauważyliśmy, sztuczne sieci neuronowe czerpią inspirację z biologicznych sieci neuronowych. Podobnie jak w biologii, sztuczne sieci neuronowe będą składać się z drobnych elementów składowych zwanych neuronami.

Z biologicznymi neuronami się zapewne kiedyś spotkałeś/aś. Nie powinno być zatem zaskoczeniem stwierdzenie, iż biologiczny neuron składa się z trzech głównych części:

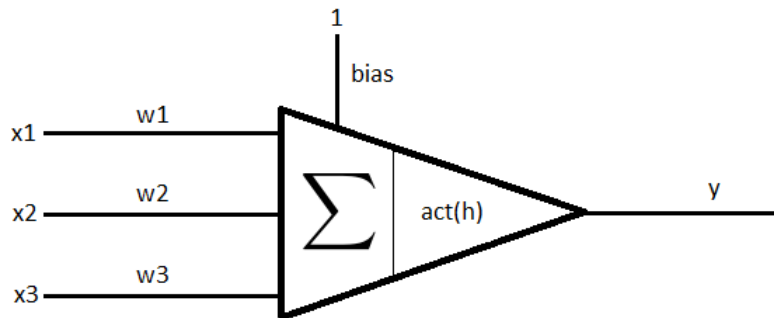
- **dendrytów** które zbierają impulsy z wyjść innych neuronów i przesyłają je do wnętrza komórki,
- **ciała komórki** gdzie następuje integracja (połączenie) sygnałów z wejść,
- **aksonu** na który wysyłane są impulsy generowane przez neuron.

Sztuczny neuron będzie w zasadzie bardzo podobny do takiego neuronu biologicznego. Również będzie posiadał **wejścia** na które będzie wysyłany sygnał z innych neuronów, **ciało neuronu** które będzie na podstawie wartości na wejściach obliczało wartość wyjścia, oraz **wyjście** neuronu które będzie się mogło dalej łączyć z wejściami kolejnych neuronów. Mimo tych podobieństw pamiętaj, iż sztuczne neurony nie są dokładnymi modelami matematycznymi prawdziwych neuronów a jedynie dość daleko idącym ich uproszczeniem! Być może najbardziej znaczącą różnicą między tymi dwoma neuronami jest fakt, iż sztuczne neurony nie działają na impulsach a na wartościach liczbowych. Wartości te możesz traktować jako coś w rodzaju częstotliwości impulsów, ale pamiętaj, że w sztucznych neuronach mogą pojawiać się wartości ujemne (podczas gdy

⁷O którym na zajęciach z ALAI nie mówimy, ale zainteresowanych zapraszamy na prowadzony przez nas fakultet *Modelowanie Procesów Sensorycznych i Poznawczych*. :)

ciężko stwierdzić, czym miałyby być ujemne częstotliwości – prawdopodobnie częstotliwościami wysyłania impulsów przez neurony hamujące).

Struktura sztucznego neuronu została przedstawiona na rysunku poniżej.



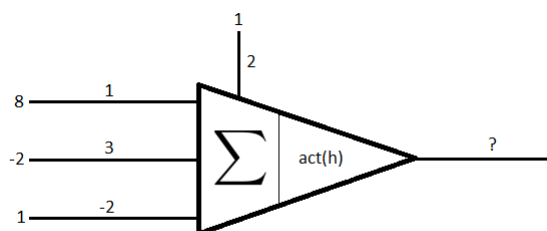
To co dotychczas sobie o nim powiedzieliśmy zostaje tutaj rozszerzone o kilka rzeczy.

Po pierwsze, z każdym wejściem neuronu związana jest pewna wartość liczbowa, którą będziemy nazywać **wagą połączenia**. Waga połączenia będzie informować neuron jak ważna jest informacja dostarczana przez dane połączenie. Waga 0 będzie oznaczała, iż dane połączenie (wejście) jest bezużyteczne. Jeżeli mamy dwa wejścia, jedno z wagą 1 i drugie z wagą 10, to będzie to oznaczać iż to drugie wejście jest 10 razy ważniejsze niż to pierwsze. Jednocześnie, waga ujemna (np. -10) będzie nam mówiła, iż owszem, dane połączenie jest ważne, ale należy zmienić znak przy wartości podawanej na to wejście. Gdy przejdziemy dalej do opisu działania neuronu, takie interpretacje wag powinny stać się w miarę oczywiste.

Po drugie, neuron będzie zawsze posiadać jedno dodatkowe wejście, na które zawsze będzie dostarczana wartość równa 1. Jak każde inne wejście, również i to będzie posiadało wagę, jako jednak, że jest to szczególna waga, będziemy ją nazywać **biasem** (ang. bias – uprzedzenie). Specjalne znaczenie tej wagi, jak również jej interpretacje, poznamy później – na razie zapamiętaj, iż taki element sztucznego neuronu istnieje.

Po trzecie, ciało sztucznego neuronu podzielone jest na dwie części. Pierwsza z nich będzie obliczała sumę ważoną wejść, podczas gdy druga (zwana **funkcją aktywacji**) będzie na podstawie owej sumy obliczała wartość na wyjściu neuronu.

A więc jak taki neuron będzie działać w praktyce? Spójrzmy na poniższy przykład:



Założmy, iż w tym przypadku funkcja aktywacji jest funkcją tożsamościową $act(h) = h$ (czyli zwraca niezmienną wartość swojego argumentu). Aby obliczyć wartość na wyjściu tego neuronu musimy:

1. Obliczyć wartość sumy ważonej wejść h . Sumę ważoną możemy obliczyć zgodnie ze wzorem

$$h = bias + \sum_{i=1}^n x_i \cdot w_i,$$

a więc w tym wypadku $h = 2 + 8 \cdot 1 + (-2) \cdot 3 + 1 \cdot (-2) = 2$.

2. Na podstawie wartości h , obliczyć wyjście z użyciem funkcji aktywacji. W tym wypadku używamy tożsamościowej funkcji aktywacji, a więc $y = act(h) = h = 2$.

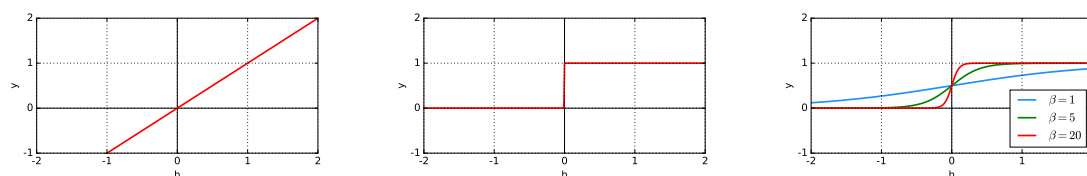
W praktyce tożsamościowa funkcja aktywacji (czy też dowolna inna liniowa funkcja aktywacji) będzie niemal zawsze bezużyteczna. Znacznie ciekawsze efekty można uzyskać dla funkcji progowej (skokowej), która zwraca 0 gdy $h < 0$ i 1 gdy $h \geq 0$. Funkcja progowa jest jednak problematyczna – nie jest ciągła i różniczkowalna w każdym punkcie, a tam gdzie jest różniczkowalna jej pochodna wynosi zero. Na chwilę obecną nie musisz rozumieć dlaczego chcemy aby funkcja aktywacji miała powyższe cechy (ani nawet co one oznaczają) – wystarczy powiedzieć, iż muszą one być spełnione aby być w stanie przeprowadzić uczenie sieci neuronowej. Samo uczenie zostanie omówione dalej.

Istnieje szereg funkcji które – choć podobne do funkcji progowej – spełniają wyżej wymienione wymagania. Jedną z nich jest **funkcja sigmoidalna unipolarna**, którą wyraża wzór

$$act(h) = \frac{1}{1 + e^{-\beta h}}$$

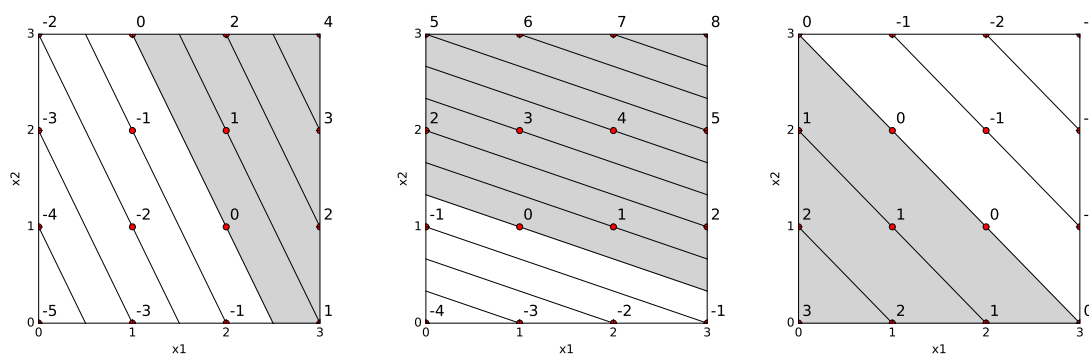
gdzie parametr β kontroluje „gwałtowność” przejścia między wartościami 0 i 1 – im wyższa wartość β tym bardziej gwałtowne przejście, gdy β rośnie do nieskończoności funkcja sigmoidalna staje się jednoznaczna z funkcją progową. To właśnie ta funkcja jest najczęściej używana w sztucznych neuronach jako funkcja aktywacji, aczkolwiek dla uproszczenia w dalszej części opracowania będziemy korzystali z funkcji progowej (o ile nie zaznaczymy inaczej).

Wykresy zależności wartości na wyjściu neuronu od wartości h dla wszystkich trzech omawianych funkcji aktywacji przedstawiono poniżej:



(a) Liniowa funkcja aktywacji (b) Progowa funkcja aktywacji (c) Sigmoidalna funkcja aktywacji

Być może myślisz sobie teraz: „Ok, zmieniliśmy funkcję aktywacji na jakąś inną, ale właściwie po co? Co taki neuron będzie robił, czego nie potrafił zrobić neuron z liniową funkcją aktywacji?”. Aby zrozumieć co właściwie będzie robił taki pojedynczy neuron, spójrz na poniższe rysunki.

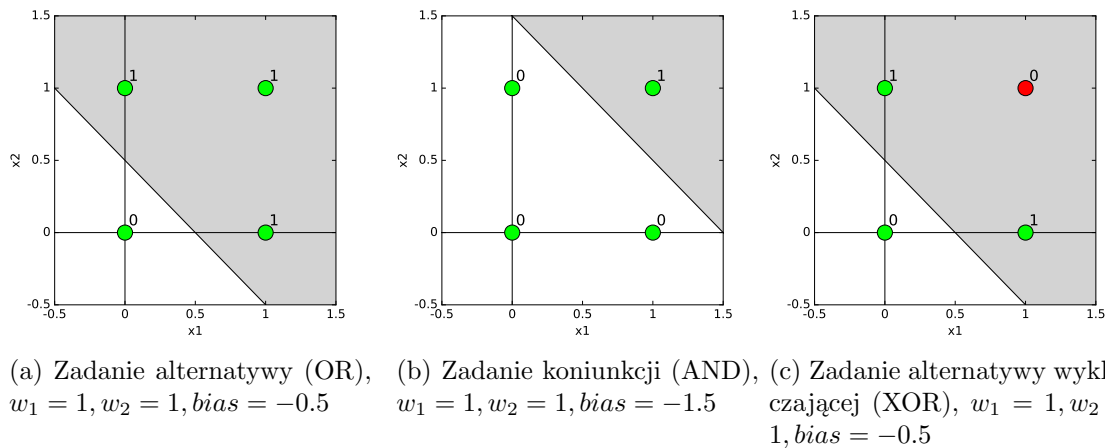


(a) $w_1 = 2, w_2 = 1, bias = -5$ (b) $w_1 = 1, w_2 = 3, bias = -4$ (c) $w_1 = -1, w_2 = -1, bias = 3$

Rysunek 12: Wykresy wartości h dla różnych wartości wag. Na osiach odpowiednio wartości x_1 i x_2 – wykres taki możemy nazwać przestrzenią wejść. Okazuje się, iż punkty o równych wartościach sumy ważonej h można połączyć prostymi równoległymi. Wartości wag przy wejściach zmieniają nachylenie prostych, wartość biasu przesuwa proste względem punktu (0,0). Neuron z progową funkcją aktywacji będzie zwracał wartość 0 dla wszystkich punktów przestrzeni po jednej stronie prostej łączącej punkty dla których $h = 0$ i 1 dla wszystkich punktów po drugiej stronie. Możemy więc powiedzieć, iż taki neuron dzieli przestrzeń wejść na dwie części.

Jak widać, pojedynczy neuron jest w stanie rozdzielić przestrzeń wejść (zbiór możliwych wejść) na dwie części. Dla dwóch wejść, przestrzeń będzie rozdzielona linią prostą (jak na po-

wyższych przykładach), dla trzech wejść – płaszczyzną, a dla więcej niż dwóch wejść – hiperpłaszczyzną (uogólnienie płaszczyzny na wyższe wymiary). Oznacza to tym samym, iż pojedynczy neuron jest już w stanie rozwiązywać pewne bardzo proste problemy klasyfikacji, takie jak na przykład koniunkcja bądź alternatywa wartości na wejściach. Nie jest on jednak jeszcze wystarczający aby rozwiązywać nieco bardziej złożone problemy (takie jak alternatywa wykluczająca). Przykłady neuronów realizujących te funkcje, zostały zamieszczone poniżej.



Rysunek 13: Wykresy zależności między wartościami wejść a wyjściem neuronu. Szara część przestrzeni odpowiada wejściom, dla których neuron zwraca wartość 1, biała odpowiada wejściom, dla których neuron zwraca 0. Wartości przy kółkach to oczekiwane (poprawne) wartości na wyjściach. Zielone kółka oznaczają poprawną klasyfikację wejść, czerwone – błędną. Jak widać, dla zadania alternatywy wykluczającej nie jesteśmy w stanie oddzielić jedną prostą wejść dla których zwracana jest wartość 1. Pojedynczy neuron jest więc niewystarczający do poprawnego zrealizowania tego zadania.

14.2 Budowa i działanie sztucznej sieci neuronowej

14.3 Uczenie sieci – backpropagation

Algorytm wstecznej propagacji błędów, który w literaturze angielskojęzycznej nosi nazwę *backpropagation algorithm* (warto znać obie nazwy), jest najbardziej popularnym algorytmem uczenia sieci neuronowych. **Algorytm backpropagation modyfikuje wyłącznie wagi połączeń między neuronami.** Architektura sieci pozostaje bez zmian.

14.3.1 Pochodne cząstkowe

„Zwykle” pochodne (tj. dotyczące funkcji jednej zmiennej) powinny być Tobie znane. W dalszej części pojawi się jednak ogólniejsze pojęcie pochodnej cząstkowej, które czujemy potrzebę intuicyjnie wyjaśnić. Jeżeli rozumiesz pochodne cząstkowe, przejdź do następnej podsekcji.

Gdy liczymy pochodne cząstkowe względem pewnej zmiennej (np. x), a nasza funkcja jest określona na większej liczbie zmiennych (np. funkcja $x + y^2 + z^3$), to traktujemy wszystkie zmienne inne niż x tak, jakby były stałymi. Jest to właściwie jedyna różnica podczas liczenia! Wszelkie znane Tobie reguły zwykłych pochodnych działają tak samo.

Poniżej znajdują się wyliczone różne pochodne cząstkowe tej samej funkcji. Jeżeli potrafisz liczyć zwykle pochodne, to pochodne cząstkowe nie powinny sprawić Tobie problemu.

$$\begin{aligned}\frac{\partial(x + y^2 + z^3)}{\partial x} &= 1 \\ \frac{\partial(x + y^2 + z^3)}{\partial y} &= 2y \\ \frac{\partial(x + y^2 + z^3)}{\partial z} &= 3z^2\end{aligned}$$

Jak widzisz, pochodne cząstkowe oznacza się symbolami ∂ (czyt. ‘de’). Na górze zawsze jest nasza funkcja, a na dole zmienna ze względu na którą różniczkujemy. Prawe strony równań powstały przez zastosowanie jednej z podstawowych reguł liczenia pochodnych: $(x^k)' = k \cdot x^{k-1}$.

Bardziej złożony przykład (czy rozumiesz, dlaczego wyszedł właśnie taki wynik?):

$$\frac{\partial(yzx + 2y^2x^2 + z^3y)}{\partial x} = yz + 4y^2x$$

14.3.2 Uczenie neuronu liniowego

Omówimy najpierw uczenie pojedynczego neuronu z liniową funkcją aktywacji. Dysponujemy zbiorem uczącym D składającym się z n par (x_j, t_j) , gdzie x_j to wejścia sieci, a t_j to pożądane wyjścia (odpowiedzi, „target”).

Błąd **pojedynczego** neuronu dla **pojedynczego przypadku uczącego j** zdefiniowany jest jako:

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}\delta^2$$

gdzie y to rzeczywista odpowiedź sieci dla wejścia x_j , a t to pożądana odpowiedź ($t = t_j$). Różnicę między t i y oznacza się jako δ (delta). Zazwyczaj E wyjdzie nam większe od 0, czyli sieć prawdopodobnie ma możliwość poprawy. Musimy zmienić jej wagi, by zminimalizować błąd. W tym celu zastosujemy heurystykę opartą na algorytmie **największego spadku gradientu** (ang. *gradient descent*). Zgodnie z nim chcemy zmienić każdą z wag w_i o:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$$

Stosujemy pochodną cząstkową po to, by oszacować udział wagi w_i w błędzie E , który jest funkcją wszystkich wag neuronu. α to **prędkość uczenia** i jest ważnym parametrem ustalonym na początku algorytmu. Mówi on o tym, jak bardzo modyfikowane będą wagi w pojedynczym kroku uczenia.

W powyższym wzorze wszystko byłoby super, tylko musimy policzyć tę pochodną cząstkową. Zgodnie z prawami pochodnych cząstkowych dla funkcji złożonych (nieco bardziej zaawansowane działanie), możemy ją rozbić na:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_i} \quad (2)$$

OK, teraz powiesz, że tylko brniemy głębiej, gdyż już mamy dwie pochodne cząstkowe do policzenia. . . Rzecz w tym, że obie są proste. :)

Najpierw obliczmy $\frac{\partial y}{\partial w_i}$. Zauważ, że y to wyjście neuronu. Przy liniowej funkcji aktywacji, jest to po prostu suma iloczynów wejść i wag, czyli $y = w_1 \cdot x_1 + \dots + w_m \cdot x_m$. Jeżeli przestudiowałeś/aś wcześniejsze przykłady pochodnych cząstkowych, to bez trudu to policzysz:

$$\frac{\partial y}{\partial w_i} = \frac{\partial(w_1 \cdot x_1 + \dots + w_m \cdot x_m)}{\partial w_i} = \frac{\partial(w_i \cdot x_i)}{\partial w_i} = x_i$$

Co do $\frac{\partial E}{\partial y}$, to rozpiszmy sobie E :

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(t^2 - 2yt + y^2) = \frac{1}{2}t^2 - yt + \frac{1}{2}y^2$$

I już widać, że policzenie z tego pochodnej będzie proste:

$$\frac{\partial E}{\partial y} = \frac{\partial(\frac{1}{2}t^2 - yt + \frac{1}{2}y^2)}{\partial y} = \frac{\partial(-yt + \frac{1}{2}y^2)}{\partial y} = -t + y = -(t - y) = -\delta$$

Gdy podstawimy to do wzoru 2, to otrzymamy:

$$\frac{\partial E}{\partial w_i} = -\delta \cdot x_i$$

Ostatecznie więc nasza zmiana wagi wyrażona jest wzorem:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \alpha \cdot \delta \cdot x_i$$

14.3.3 Uogólnione uczenie neuronu

W powyższym przykładzie założyliśmy, że neuron ma liniową funkcję aktywacji. Chcielibyśmy jednak uogólnić powyższe rozważania na dowolną funkcję aktywacji (liniowe neurony nie są interesujące). Okazuje się, że można to dosyć prosto zrobić. Wróćmy do równania 2:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_i}$$

y jest wartością zwracaną przez neuron. Wartość ta powstaje przez zastosowanie funkcji aktywacji do sumy iloczynów wejść i wag. Neuron liniowy po prostu propaguje tę sumę dalej. By uwzględnić dowolną (różniczkowalną) funkcję aktywacji musimy dodać do wzoru pewną pośrednią wartość h , która reprezentuje sumę iloczynów wejść i wag ($h = x_1 w_1 + \dots + x_n w_n$), ponieważ funkcja aktywacji przyjmuje jako argument właśnie tę sumę ($y = act(h)$).

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w_i} = \frac{\partial y}{\partial h} \cdot \frac{\partial(w_1 \cdot x_1 + \dots + w_m \cdot x_m)}{\partial w_i} = \frac{\partial y}{\partial h} \cdot \frac{\partial(w_i \cdot x_i)}{\partial w_i} = \frac{\partial y}{\partial h} \cdot x_i$$

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \alpha \cdot \delta \cdot \frac{\partial y}{\partial h} \cdot x_i \quad (3)$$

Liniowa funkcja aktywacji wyrażona jako funkcja h :

$$y = act(h) = h$$

Pochodna po h z takiej funkcji będzie równa 1, i dlatego właśnie ten człon „zniknął” w przykładzie w poprzedniej sekcji. Wyrażenie $\frac{\partial y}{\partial h}$ można zapisać równoważnie jako $act'(h)$.

Równanie 3 nazywane jest **regułą delta** i określa ono, jak bardzo zmieni się dana waga neuronu.

14.3.4 Backpropagation

Omówimy teraz algorytm *wstecznej propagacji błędów* używany do uczenia wielowarstwowych sieci neuronowych. Wykorzystuje on *regułę delta* do „lokalnego” (to znaczy niewuwzględniającego poprawek dokonywanych na sąsiednich neuronach z warstwy oraz neuronach z innych warstw) uczenia pojedynczych neuronów.

Neurony z warstwy wyjściowej możemy łatwo nauczyć korzystając bezpośrednio z *reguły delta* (wzór 3), ponieważ mamy wyjście y całej sieci i możemy je porównać z oczekiwaną wartością t . Problemy zaczynają się jednak z neuronami w warstwach ukrytych, ponieważ nie wiemy jak wyznaczyć ich błąd (δ). Wykazano, że jeżeli sieć jako całość ma stosować się do reguły największego spadku gradientu, to za błąd δ neuronu N w warstwie ukrytej powinniśmy przyjąć **sumę ważoną błędów neuronów w następnej warstwie, do których neuron N przesłał**

swoją aktywację. Suma ta ważona jest wagami połączeń między neuronami – intuicja jest taka, że jeżeli neuron N przesłał aktywację do jakiegoś neuronu M w następnej warstwie z dużą wagą, to miał tym samym duży udział w ewentualnym błędzie popełnionym przez tamten neuron. W ten sposób błąd jest niejako przekazywany „w tył” od warstwy wyjściowej do wejściowej, i stąd taka nazwa tego algorytmu.

Zauważ, że błąd δ neuronu w warstwie ukrytej był jedynym komponentem z reguły delta, którego nam brakowało do jej zastosowania. W momencie, gdy przyjęliśmy opisaną wyżej strategię jego wstecznej propagacji, możemy wykorzystać regułę delta analogicznie jak poprzednio dla jednego neuronu.

14.3.5 Przebieg uczenia

Na początku sieć jest inicjalizowana losowymi wagami. Przypadki uczące podawane są kolejno, zazwyczaj w porządku losowym, na wejście sieci. Aktualizacja wag sieci algorytmem back-propagation następuje dla każdego przypadku uczącego po obliczeniu dla niego wyjścia sieci. Pełen cykl prezentacji wszystkich przykładów uczących nazywany jest **epoką**.

Po epoce zazwyczaj sprawdza się aktualny stan błędu na osobno wydzielonym zbiorze walidacyjnym. Jeżeli widzimy, że trafność klasyfikacji zaczyna na nim znacząco spadać, to przerywamy uczenie by uniknąć przeuczenia.

15 Słowa końcowe



A tak na serio, to powodzenia! :) Mamy nadzieję, że ten dokument pomógł w zrozumieniu zagadnień, oraz że zdadzą wszyscy. Uczenie maszynowe, podobnie jak większość rzeczy wymyślonych przez człowieka, tak naprawdę u podstaw jest proste. Po prostu nie wszyscy jeszcze to widzą...