2021

Metody Sztucznej Inteligencji. 5. Uczenie się klasyfikacji i aproksymacji funkcji

5. UCZENIE SIĘ KLASYFIKACJI I APROKSYMACJI FUNKCJI

WŁODZIMIERZ KASPRZAK

UCZENIE PRZEZ OBSERWACJĘ, TWORZENIE POJĘĆ - GRUPOWANIE, DRZEWO DECYZJI, KLASYFIKATOR K-NN, KLASYFIKATOR SVM, METODY REGRESJI, LSE, NIELINIOWA LSE

Przedstawione zostanie pojęcie uczenia się przez obserwację (indukcję). Jego rodzajami są: tworzenie pojęć (grupowanie), uczenie się rozróżniania pojęć (klasyfikacji) i uczenie się aproksymacji funkcji. Jako przykłady tworzenia pojęć przedstawione zostaną algorytmy „k-mean" i EM służące do tworzenia klastrów w przestrzeni cech. Pojęcia zdefiniowane za pomocą własności (atrybutów) można rozróżniać dzięki klasyfikatorom zdefiniowanym według drzew decyzyjnych i lasów losowych. Typowe klasyfikatory oparte o funkcje potencjału (gęstości) w przestrzeni cech to klasyfikator Bayesa lub klasyfikator k najbliższych sąsiadów. Popularnym klasyfikatorem dyskryminacyjnym w przestrzeni cech jest „maszyna wektorów nośnych" SVM. Przedstawione zostanie uczenie się aproksymacji funkcji poprzez regresję liniową i nieliniową.

Spis treści

1 Uczenie się na podstawie obserwacji

1.1 Indukcja

1.1.1 Pojęcie (*concept*)

Uczenie się przez indukcję polega na wnioskowaniu (nabywaniu wiedzy) na podstawie obserwacji i zakłada się w nim dostępność próbek uczących (inna nazwa to „przykłady trenujące"):

$$T \Rightarrow h \wedge W ,$$

gdzie h – hipoteza wygenerowana w wyniku wnioskowania indukcyjnego, W - wiedza wrodzona, T – przykłady trenujące (próbki uczące), $\Rightarrow$ - logiczna implikacja (h, W, T są prawdziwe).
Indukcyjna hipoteza wraz z wiedzą wrodzoną (być może pustą) wyjaśnia możliwie dokładnie otrzymaną informację uczącą. Celem powyższego wnioskowania jest odkrycie pewnych zależności w danych i wyjaśnienie przykładów trenujących oraz przede wszystkim przewidywanie nowych faktów i obserwacji. Założenie o indukcji mówi o tym, że hipoteza wybrana na podstawie zbioru uczącego sprawdza się także dla przykładów spoza tego zbioru.

Pojęcia są formą reprezentacji wiedzy. Pojęcie przypisuje *obiektom* lub *zdarzeniom* etykiety ich klasy. Przez „pojęcie" rozumiemy podzbiór obiektów lub zdarzeń definiowanych w ramach pewnego większego zbioru – „kategorii". Np. pojęcie „ptak" jest podzbiorem obiektów w zbiorze wszystkich rzeczy takich, które zaliczamy do kategorii „ptaków".
Formalnie „pojęcie" oznacza istnienie takiej funkcji charakterystycznej zbioru, która przyjmuje wartość „True" (1) dla przykładu należącego do pojęcia i „False" (0) w przeciwnym przypadku:

$$c(x) = \begin{cases} 1 & \text{gdy przykład x należy do kategorii (przykład pozytywny)} \\ 0 & \text{w przeciwnym przypadku (przykład negatywny)} \end{cases}$$

Oznaczmy: T - zbiór przykładów trenujących pochodzących z pewnej dziedziny X. Wiedza zdobyta na ograniczonym zbiorze przykładów T ma być użyteczna do przewidywania właściwości przykładów x z całej dziedziny X. Założymy, że dziedzina jest opisana za pomocą ustalonego zbioru atrybutów:
$A = \{a_1, a_2, \ldots, a_n\}$, gdzie

$$a_1 : X \to A_1, \quad a_2 : X \to A_2, \quad \ldots, \quad a_n : X \to A_n.$$

Przykład (próbkę) utożsamimy z wektorem wartości atrybutów:

$$\{x \in X : x = [a_1(x) = v_1, a_2(x) = v_2, \ldots, a_n(x) = v_n,].$$

1.1.2 Rodzaje indukcji

Wyróżnimy trzy podstawowe sposoby indukcyjnego uczenia:

1. **tworzenie pojęć** - proces wyodrębniania pojęć (klas w wyniku nienadzorowanego uczenia),
2. **uczenie się pojęć** - uczenie się sposobu przydzielania obiektów do klas - uczenie się **klasyfikacji** (proces nadzorowanego uczenia);
3. **uczenie się aproksymacji funkcji** - uczenie się odwzorowywania obiektów na liczby rzeczywiste (proces nadzorowanego uczenia).

Tworzenie pojęć. Niech próbki (przykłady) uczące pozbawione są etykiety klasy, do której należą. W procesie tworzenia pojęć należy znaleźć zależności (podobieństwa cech) pomiędzy przykładami i pogrupować przykłady w klasy. **Grupowanie** (klasteryzacja) odbywa się na zasadzie minimalizowania różnic przykładów z jednej grupy i maksymalizowania różnić pomiędzy grupami.
Uczenie się pojęć polega na budowie pewnej hipotezy *h*, która na podstawie wartości atrybutów przykładu potrafi estymować związane z nim pojęcie *C* :

$$h : X \mapsto C$$

Najczęściej hipoteza jedynie przybliża rzeczywiste kategorie w lepszym lub gorszym stopniu:

$$h(x) \approx c(x)$$

Uczenie polega na znalezieniu hipotezy, która możliwie najlepiej odzwierciedla pojęcie docelowe w zbiorze etykietowanych przykładów *T*. Wybór hipotezy ma minimalizować błąd rzeczywisty na całej dziedzinie, który estymuje się za pomocą błędu na zbiorze trenującym.
Uczenie się aproksymacji funkcji jest podobne do uczenia się pojęć z tą różnicą, że liczba kategorii jest zbiorem liczb rzeczywistych, a nie określonym z góry skończonym zbiorem jak w przypadku uczenia się pojęć.

- Próbka ucząca ma postać sekwencji argumentów funkcji oraz wartości funkcji dla tych argumentów, która odpowiada etykiecie kategorii.
- Celem uczenia jest aproksymacja postaci funkcji na podstawie przykładów na całą dziedzinę.

1.1.3 Tryb podawania próbek

Algorytmy indukcyjnego uczenia się można implementować jako strategie przeszukiwania przestrzeni hipotez:

1. poszukujemy dobrej hipotezy w pewnej przestrzeni możliwych pojęć czy funkcji, zdefiniowanej w języku wybranym dla tego zadania;
2. przestrzeń hipotez powinna zawierać wszystkie hipotezy jakie może skonstruować uczeń;
3. przynajmniej jedna z tych hipotez jest poprawna.

Hipoteza jest **nadmiernie dopasowana** do zbioru uczącego, jeśli inna hipoteza o większym błędzie próbki na tym zbiorze ma mimo to mniejszy błąd rzeczywisty. Aby ograniczyć ryzyko nadmiernego dopasowania należy preferować hipotezy proste o małym błędzie próbki na zbiorze uczącym. Można ograniczyć liczbę pomyłek stosując odpowiedni *tryb podawania* próbek uczących.
Wyróżnimy następujące tryby podawania próbek:

1. wsadowy
2. inkrementacyjny
3. epokowy

W trybie wsadowym cały zbiór trenujący jest od razu dostępny, a po jego przetworzeniu podawana jest hipoteza. Powiększenie zbioru trenującego (pojawienie się nowych przykładów) powoduje konieczność rozpoczęcia uczenia się od nowa.

W trybie inkrementacyjnym uczeń otrzymuje na raz tylko jeden przykład i zgodnie z nim zmienia swoją hipotezę. W każdej chwili uczenia dostępna jest hipoteza uznawana za ostateczną, gdy wyczerpią się próbki.
W trybie epokowym proces uczenia zorganizowany jest w cykle zwane epokami, w każdej z których jest przetwarzany pewien zbiór przykładów trenujących. Gdy epoka składa się jedynie z jednej próbki to otrzymujemy tryb inkrementacyjny.

2 Tworzenie pojęć - uczenie bez nadzoru

Tworzenie pojęć może polegać na grupowania (klasteryzacji) próbek uczących pozbawionych etykiet klas. Jest to problem uczenia bez nadzoru - przykład samoorganizacji systemu agentowego.

2.1 Klasteryzacja "*k*-średnich"

Jest to podstawowy algorytm grupowania i organizacji danych. Dlatego stanowi on nasz punkt wstępu do tych zagadnień. Zakładamy, że istnieje pewna liczba obserwacji o postaci wektorów cech, c_j (j=1,2 ..., n). Niech spodziewana liczba klas wynosi K. Algorytm k-średnich organizuje przekazane mu dane w k klastrów (Tabela 1).

Tabela 1. Algorytm grupowania k-średnich (k-means)

1. Inicjalizuj dowolnie K środków klastrów: $\mu^{(i)}$.
2. FOR każda obserwacja c_j DO
   przypisz go do najbliższego klastra, tzn.
   ustaw $\xi(c_j) \leftarrow i$, wtedy gdy $\mathrm{d}(\mu^{(i)}, c_j) = \min_k \mathrm{d}(\mu^{(k)}, c_j)$,
   gdzie d(,) jest pewną miarą odległości (zwykle miarą Euklidesa).
3. FOR każda klasa DO
   wyznacz wektor średni obserwacji przypisanych do danej klasy
$$\mu^{(i)} = \frac{1}{n^{(i)}} \sum_{j,\xi(c_j)=i} c_j$$
4. Powtarzaj kroki 2 i 3 zadaną liczbę iteracji.

Podstawowym problemem projektowym dla algorytmu *k*-średnich jest właściwa inicjalizacja środków klas. Najczęściej wykonuje się go wielokrotnie dla różnych wartości początkowych i wybiera najlepszy wynik. Wadą algorytmu *k*-średnich jest brak gwarancji zbieżności do stanu ustalonego

2.2 Klasteryzacja EM ("Expectation Maximization")

Metoda EM (ang. expectation-maximization) jest ogólnym podejściem statystycznym do radzenia sobie z brakującą informacją. Możemy ją zastosować do stochastycznego grupowania danych w postaci mieszaniny rozkładów Gaussa (GMM – *Gaussian mixture model*). Wtedy dostarcza ona

oszacowanie największej wiarygodności (ML – *maximum likelihood*) dla parametrów mieszaniny *k* rozkładów Gaussa:

$$p(c|\theta) = \sum_{j=1}^{K} \alpha^j N(c|\mu^j, \Lambda^j) \quad (5.1)$$

gdzie *N*(.) oznacza funkcję Gaussa gęstości prawdopodobieństwa (tzw. rozkład normalny). Każdy klaster *j* (=1, ..., *K*) charakteryzowany jest rozkładem Gaussa o parametrach (μ^j, Λ^j). Te parametry i wagi rozkładów, α^j , są tak dobrane w procesie klasteryzacji EM, aby maksymalizować sumę prawdopodobieństw (5.1) liczoną po wszystkich próbkach uczących.

Poza inicjalizacją, algorytm klasteryzacji EM składa się z dwóch kroków wykonywanych iteracyjnie, aż do spełnienia warunku stopu (Tabela 2):

- krok E generuje aktualną "miękką" klasteryzację, tzn. wartość P^i_j oznacza prawdopodobieństwo przynależności do klastra *i* próbki c_j ;
- krok M jest aktualnym oszacowaniem wszystkich parametrów rozkładu: α^i, μ^i, Λ^i .

Tabela 2. Klasteryzacja EM dla modelu GMM

Dane: obserwacje (wektory cech) c_j (j=1,2 ..., *n*), liczba klas wynosi K.

1. Inicjalizuj parametry: $\theta = \{ \alpha^i, \mu^i, \Lambda^i / i=1,\ldots,K \}$.

2. Powtarzaj sekwencję kroków E i M oraz sprawdzanie warunku zatrzymania:

- Krok E:

FOR każda próbka c_j DO

FOR każda klasa *i* DO

$$P_j^i = \frac{p(c_j \mid \xi(c_j) = i, \theta)\alpha^i}{\sum_i p(c_j \mid \xi(c_j) = i, \theta)\alpha^i} \quad (5.2)$$

- Krok M:

FOR każda klasa *i* DO

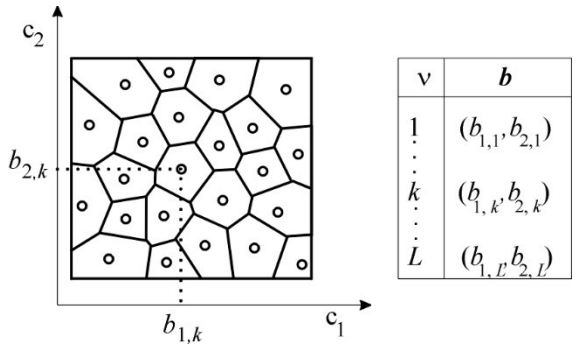
$$\mu^i = \frac{\sum_j c_j P_j^i}{\sum_j P_j^i}, \qquad \Lambda^i = \frac{\sum_j (c_j - \mu^i)(c_j - \mu^i)^T P_j^i}{\sum_j P_j^i}, \qquad \alpha^i = \frac{\sum_j P_j^i}{\sum_i \sum_j P_j^i} \quad (5.3)$$

- Powtarzaj dopóki nie wykonano zadanej z góry liczby iteracji.

2.3 Wyznaczanie liczby klastrów

2.3.1 Kwantyzacja wektorowa dla *K* klas

Algorytmy *kwantyzacji wektorowej* często stosują metodę „*k*-średnich" dla wyznaczenia podziału przestrzeni reprezentacji na podobszary odpowiadające klastrom i ich reprezentowania przez ich wektory średnie. W ten sposób w procesie uczenia wyznaczany jest słownik kodowy złożony z reprezentantów *K* klas. Następnie na etapie kodowania *kwantyzator* zastępuje wektor cech przez indeks reprezentanta w słowniku kodowym (Rysunek 1).



Rysunek 1. Idea 2-wymiarowej kwantyzacji wektorowej: klasteryzacja przestrzeni cech i przypisanie kodów ze słownika kodowego reprezentantom klastrów.

Zazwyczaj kwantyzator rozszerza ideę klasteryzacji *k*-średnich o adaptacyjny warunek stopu. W podanym poniżej algorytmie (Tabela 3) w każdej iteracji badana jest zbieżność aktualnego błędu kwantyzacji próbek uczących. Jeśli błąd przestaje się istotnie zmniejszać iteracja jest przerywana.

Tabela 3. Klasteryzacja i kwantyzacja wektorowa

Dane są:

- zbiór wektorów cech – próbek uczących, $\omega = \{c^i \mid i = 1,..., N\}$,
- liczba klas *K*,
- początkowy słownik $Z^{(0)}$ złożony z wektorów prototypów, $Z^{(0)}=\{z^{(0)}_\kappa, \kappa=1,...,K\}$,
- początkowy błąd aproksymacji $\varepsilon^{(0)}$,
- zadany próg Θ dla względnego błędu aproksymacji.

1. Iteruj kroki (a)-(d) dla $I = 1,2,3, \ldots$

(a) zaklasyfikuj wektory próbek $c^i \in \omega$ zgodnie z aktualnym słownikiem kodowym $Z^{(I-1)}$ – wybierając klasę odpowiadającą najmniejszej odległości próbki od prototypu klasy – i wyznacz sumaryczny błąd klasyfikacji

$$\varepsilon^{(I)} = \sum_{i=1}^{N} \min_\kappa \mid \mathbf{c}_i - z_\kappa^{(I-1)} \mid . \tag{5.4}$$

(b) jeśli względna różnica błędów jest poniżej zadanego progu, tzn.

$$\frac{\mid \varepsilon^{(I)} - \varepsilon^{(I-1)} \mid}{\varepsilon^{(I-1)}} < \Theta \tag{5.5}$$

to przejdź do kroku KONIEC.

(c) oblicz nowy słownik kodowy $Z^{(I)}$ z prototypami klas $z^{(I)}_\kappa$:

$$\mathbf{z}_k^{(I)} = \frac{1}{N_k^{(I)}} \sum_{c \in \Omega_k^{(I)}} \mathbf{c} \tag{5.6}$$

(d) ustaw $I = I+1$ i ponów iterację od kroku (a).

(KONIEC) Znaleziony słownik kodowy to $Z^{(I)}$ przy błędzie aproksymacji $\varepsilon^{(I)}$.

Jako wynik działania algorytmu kwantyzacji otrzymuje się zbiór klastrów i przydzielonych do nich reprezentantów (punkty centralne w każdym klastrze, określane mianem centroidów), spełniających następujące warunki:

1. Każdy punkt należy do klastra, którego centroid jest położony najbliżej tego punktu.
2. Każdy centroid jest punktem, który jest najmniej oddalony od wszystkich pozostałych punktów klastra (suma odległości tego punktu od wszystkich pozostałych jest najmniejsza, wg przyjętej miary odległości, np. Euklidesa).

2.3.2 Pełna kwantyzacja wektorowa (z wyznaczaniem liczby klas)

Pełna kwantyzacja polega na nienadzorowanym uczeniu zarówno **liczby** klas jak i **rozkładu** klastrów w przestrzeni cech ().

Tabela 4. Kwantyzacja i wyznaczanie liczby klastrów

DANE wejściowe:

- zbiór wektorów cech – próbek uczących: $\omega = \{c_i \mid i = 1,..., N\}$,
- próg dla minimalnego błędu Γ.

1) Oblicz centroidę całego zbioru próbek ω :

$$\boldsymbol{\mu}^{(0)} = \frac{1}{N}\sum_i \mathbf{c}_i \tag{5.7}$$

2) Ustaw początkowy słownik $Z^{(0)}$ dla K_0= 2 klas i posiadających prototypy:

$$z_{1,2} = (1 \pm \delta)\,\mu \text{ , gdzie } \delta < 1. \tag{5.8}$$

3) Iteruj kroki (4)-(7) dla $K = K_0$, K_0 +1 , K_0 +2, …

4) Wykonaj *kwantyzację wektorową k-średnich* dla (K, $Z^{(0)}$). Powstaje nowy słownik $Z^{(I)}$ obarczony błędem $\varepsilon^{(I)}$.

5) IF ($\varepsilon^{(I)} < \Gamma$) THEN STOP.

6) Ustaw nowy słownik $Z^{(0)}$ dla K + 1 klas o prototypach:

- wybierz klaster K_κ o największej wariancji próbek tej klasy i podziel klaster na 2 części o centroidach:

$$z_{\kappa,K+\kappa} = (1 \pm \delta)\, z_\kappa \text{ , gdzie } \delta < 1. \tag{5.9}$$

- pozostałe klastry przechodzą bez zmian do nowego zbioru klastrów.

7) Przejdź do (3) i wykonuj następną iterację kroków (4-7).

2.4 Sieć neuronowa Kohonena

Sieć Kohonena jest 2-warstwową siecią neuronową. W pierwszej warstwie każde z wyjść i jest połączone z każdym wejściem j, zaś funkcja i-tego neuronu wynosi:

$$y_i = \mathbf{w}_i^{\mathrm{T}}\,\mathbf{x}\,.$$

Wektor $\mathbf{w}_i$ jest reprezentantem i-tego klastra. Tym samym wyjście y_i jest iloczynem skalarnym wektora wejściowego i reprezentanta klastra. Często przyjmuje się, że zarówno wektory wag jak i

dane wejściowe są znormalizowane – mają jednostkowy moduł. Jeśli tak nie jest, to aby uzyskać znormalizowany współczynnik korelacji stosuje się normalizację wyjścia: $z_i = f(y_i) = \frac{y_i}{|x|\ |w_i|}$.

W drugiej warstwie następuje wybór jednego neuronu wyjściowego *k* o najwyższej wartości funkcji aktywacji. Odbywa się to na drodze iteracyjnej modyfikacji wartości aktywacji neuronów, aż do chwili, gdy $z_k \approx 1$ i $z_i \approx 0$, dla $i \neq k$. W tej warstwie każdy *i*-ty neuron i jest połączony z innymi *m* łukami pobudzającymi, zaś ze sobą samym łukiem osłabiającym pobudzenie:

$$v_{im} = \begin{cases} -\varepsilon & dla\ i \neq m \\ 1 & dla\ i = m \end{cases}$$

Wagi pierwszej warstwy sieci Kohonena uczone są metodą nienadzorowanego uczenia „w warunkach konkurencji" (ang. *competitive learning*). Jest to realizacja grupowania (klasteryzacji) danych. Idea algorytmu uczenia w warunkach konkurencji jest następująca (Tabela 5):

- wybierany jest neuron wyjściowy o najwyższej aktywacji i jego wagi $\mathbf{w}_l$ są korygowane „w kierunku" aktualnego wektora wejściowego;
- Jednocześnie od wektora wejściowego „odsuwana" jest najbliższa z pozostałych waga $\mathbf{w}_k$.

Tabela 5. Uczenie w warunkach konkurencji

Niech *N* – zadana liczba neuronów wyjściowych $\mathbf{w}_i$, $i=1,\ldots,N$ Z założenia wektory wag i próbki uczące { **x** } są jednostkowe. FOR (dla próbki **x**(*t*)) DO (1) Po określeniu wartości pobudzenia wyjść znajdujemy dwóch zwycięzców - *k* i *l* - stosując miarę odległości wejścia **x** od wag: $\|\mathbf{x}(t) - \mathbf{w}_k\| < \|\mathbf{x}(t) - \mathbf{w}_l\| < \|\mathbf{x}(t) - \mathbf{w}_i\|$, . (2) Przesuwamy wektor wag najbliższy wejściu w kierunku wejścia: $\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\ [\mathbf{x}(t) - \mathbf{w}_k(t)]$ (3) Drugi najbliższy wektor wag ewentualnie „odsuwamy" od wektora **x**(*t*): $\mathbf{w}_l(t+1) = \mathbf{w}_l(t) - \alpha\ [\mathbf{x}(t) - \mathbf{w}_l(t)]$

3 Uczenie się klasyfikacji

3.1 Zadanie klasyfikacji

Zadanie klasyfikacji polega na wyznaczeniu pojęcia (klasy) dla zadanego przykładu (obiektu obserwacji). Klasyfikator musi być najpierw nauczony. Przykłady trenujące (próbki uczące) dla klasyfikatora mają postać par:

- opis obiektu i
- etykieta jego klasy.

Taką parę nazywamy przykładem etykietowanym. Przykładem nieetykietowanym jest taki, który obejmuje jedynie opis obiektu.

Klasę pojęć nazywamy **nauczalną**, jeżeli istnieje efektywny algorytm znajdujący z dużym prawdopodobieństwem pojęcie „w przybliżeniu poprawne" – co oznacza, że pojęcie to poprawnie

obejmuje duży procent możliwych przykładów klasy. Zasada na której opiera się całe rozumowanie to skrócie: hipoteza zgodna z wystarczająco dużym zbiorem trenującym jest prawdopodobnie w przybliżeniu poprawna.

3.2 Drzewo decyzyjne w klasyfikacji

W drzewie decyzyjnym występują:

- **węzły** niekońcowe – odpowiadają one testom przeprowadzanym na atrybutach przykładów,
- **gałęzie** (krawędzie) – odpowiadają one możliwym wynikom tych testów,
- **liście** (węzły końcowe) – odpowiadają one etykietom klas.

Klasyfikacja przykładu za pomocą drzewa decyzyjnego polega na przejściu ścieżki od korzenia do liścia drzewa wzdłuż gałęzi wyznaczanych przez wyniki testów związanych z odwiedzanymi kolejno węzłami. Osiągnięcie liścia wyznacza klasę dla tego przykładu – drzewo decyzyjne reprezentuje **hipotezę**.

3.2.1 Przykład 1

Nasz problem decyzyjny może polegać na tym, czy czekać na wolny stolik w restauracji czy nie czekać i wyjść? Załóżmy, że dla podjęcia decyzji korzystamy z następujących **atrybutów**:

- **Alternatywa**: czy jest inna restauracja w okolicy?
- **Bar**: czy można oczekiwać w barze?
- **Piątek**: czy dziś jest piątek czy już sobota?
- **Głód**: czy jesteśmy głodni?
- **Osoby**: liczba osób w restauracji (nikogo, trochę, pełno)
- **Cena**: zakres cenowy (+, ++, +++)
- **Deszcz**: czy pada deszcz?
- **Rezerwacja**: czy mamy rezerwację?
- **Typ**: rodzaj restauracji (francuska, włoska, tajlandzka, *burger*)
- **Czas**: szacowany czas czekania (0-10 min.,10-30, 30-60, >60).

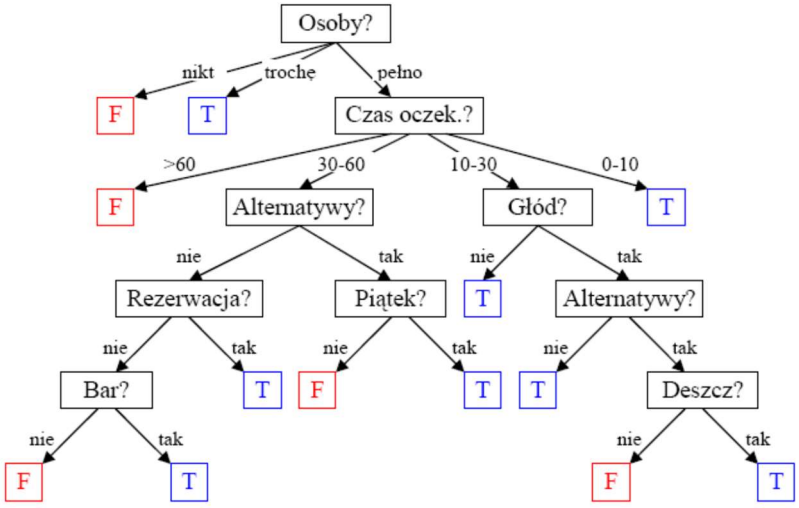
Próbki uczące (przykłady trenujące) są opisane poprzez wartości atrybutów, które w naszym przykładzie mogą być typu logicznego, dyskretne (liczby całkowite) lub ciągłe. Na potrzeby drzewa decyzyjnego ciągła dziedzina zostaje podzielona na przedziały. Każdy przykład (decyzja) należy do jednej z dwóch klas (Tabela 6):

- pozytywną – decyzja „*czekać*" (T) lub
- negatywną – decyzja „*nie czekać*" (F).

Tabela 6. Próbki uczące dla problemu „czekania w restauracji”

Próbka	Atrybuty										CEL
	Alt	Bar	Piąt	Głód	Osoby	Cena	Deszcz	Rez	Typ	Czas	Czekaj
X1	Tak	Nie	Nie	Tak	Trochę	+++	Nie	Tak	franc	0-10	True
X2	Tak	Nie	Nie	Tak	Pełno	+	Nie	Nie	tajski	30-60	False
X3	Nie	Tak	Nie	Nie	Trochę	+	Nie	Nie	burger	0-10	True
X4	Tak	Nie	Tak	Tak	Pełno	+++	Nie	Nie	tajski	10-30	True
X5	Tak	Nie	Tak	Nie	Pełno	+++	Nie	Tak	franc	>60	False
X6	Nie	Tak	Nie	Tak	Trochę	++	Tak	Tak	włoski	0-10	True
X7	Nie	Tak	Nie	Nie	Nikt	+	Tak	Nie	burger	0-10	False
X8	Nie	Nie	Nie	Tak	Trochę	++	Tak	Tak	tajski	0-10	True
X9	Nie	Tak	Tak	Nie	Pełno	+	Tak	Nie	burger	>60	False
X10	Tak	Tak	Tak	Tak	Pełno	+++	Nie	Tak	włoski	10-30	False
X11	Nie	Nie	Nie	Nie	Nikt	+	Nie	Nie	tajski	0-10	False
X12	Tak	Tak	Tak	Tak	Pełno	+	Nie	Nie	burger	30-60	True

Przykład drzewa decyzyjnego, który powstaje w tym problemie, pokazano na Rysunek 1.



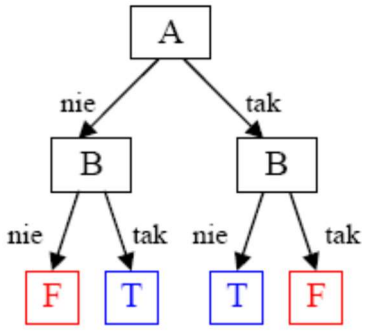
Rysunek 2. Przykład drzewa decyzyjnego dla problemu „czekanie w restauracji”.

3.2.2 Siła wyrazu drzew decyzyjnych

Drzewa decyzyjne mogą wyrazić każdą **funkcję logiczną** lub **dyskretną** zależną od zbioru atrybutów wejściowych. Np. dla funkcji logicznych (o wartościach Boole’a) pojedynczy wiersz w tabeli prawdy odpowiada pojedynczej ścieżce w drzewie od korzenia do liścia (Rysunek 3).

A	B	A ⊗ B
F	F	F
F	T	T
T	F	T
T	T	F

⊗ oznacza funkcję „XOR”



Rysunek 3. Funkcja logiczna (lewa strona) i jej reprezentacja w postaci drzewa decyzyjnego (prawa strona).

Jaka jest liczba możliwych drzew decyzyjnych dla *n* atrybutów typu logicznego (Boolean)? Każda funkcja logiczna o *n* binarnych atrybutach odpowiada tablicy prawdy o 2^n wierszach. Liczba różnych tablic prawdy o takim rozmiarze wynosi 2^{2n}. Np. dla n=6 istnieje 18,446,744,073,709,551,616 drzew. Tak duża przestrzeń problemu uczenia wymaga stosowania nietrywialnych algorytmów uczenia drzew decyzyjnych.

3.2.3 Uczenie się drzewa decyzyjnego

Dla deterministycznej funkcji zmiennych zawsze można utworzyć drzewo decyzyjne w pełni zgodne z próbkami uczącymi takie, że każdej próbce odpowiadać będzie pojedyncza ścieżka od korzenia do liścia. Jednak drzewo to zwykle nie będzie efektywnie uogólniać (przewidywać) nieznanych próbek. Interesuje nas wyznaczenie takiego drzewa decyzyjnego, które jest możliwie zbalansowane tak, aby decyzja mogła zapaść jak najszybciej dla próbek z całej przestrzeni cech. Idea metody uczenia drzewa decyzyjnego (Tabela 7):

1. Cel – szukamy zwartego, dobrze zbalansowanego drzewa, zgodnego z próbkami uczącymi.
2. Postępowanie – rekursywnie wybieramy "najważniejszy" atrybut do roli korzenia pod-drzewa. Dla określenia „ważności” atrybutu zastosujemy kryteria znane z teorii informacji.

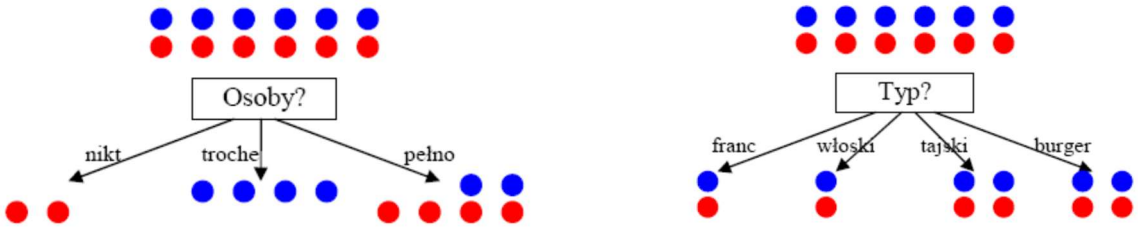
Tabela 7. Algorytm uczenia się drzewa decyzyjnego

```
function DTL(próbki, atrybuty, domyślna) returns drzewo decyzyjne
{ if (próbki == ∅) then return domyślna;
    else if (wszystkie próbki należą do jednej klasy)
        then return KLASA(próbki);
        else if (atrybuty == ∅)
            then return KLASAWIĘKSZOŚCIOWA(próbki);
            else
            {   best ← WYBIERZATRYBUT(atrybuty, próbki);
                drzewo ← nowe drzewo o korzeniu testującym atrybut best ;
                m ← KLASAWIĘKSZOŚCIOWA(próbki);
                for each (wartość v_i ∈ best) do
                {   próbki_i ← {elementy w próbki o best = v_i};
                    poddrzewo ← DTL(próbki_i, atrybuty - best, m);
                    dodaj gałąź do drzewa o etykiecie v_i i poddrzewie poddrzewo;
                }
            }
            return drzewo;
}
```

W podfunkcji **„WybierzAtrybut”** algorytmu DTL następuje wybór takiego atrybutu, który najlepiej dzieli próbki uczące na podzbiory dwóch klas – w idealnym przypadku na takie dwa podzbiory, gdzie jeden zawiera “wszystkie pozytywne” próbki a drugi “wszystkie negatywne”.

Przykład 2

Dla próbek z przykładu 1 testując atrybut *Osoby* (Rysunek 4, lewa strona) zbliżamy się bliżej do celu rozdzielenia przypadków pozytywnych od negatywnych. Testowanie atrybutu *Typ* (Rysunek 4, prawa strona) zasadniczo niczego w tym sensie nie poprawia.



Rysunek 4. Alternatywne drzewa tworzone przy testowaniu dwóch różnych atrybutów

3.2.4 Kryteria wyboru testów

Formalnie biorąc implementacja podfunkcji **WybierzAtrybut** oparta może być na teorii informacji. Oczekiwana wartość informacji (**entropia**) dla zmiennej losowej X o n wartościach wynosi:

$$H(P_X(v_1), \ldots, P_X(v_n)) = - \Sigma_{i=1}^{n} [P_X(v_i) \log_2 P_X(v_i)]. \quad (5\text{-}10)$$

Im bardziej prawdopodobna jest realizacja zmiennej losowej tym mniej informacji ona zawiera. Z kolei im większa niepewność zdarzenia tym więcej zawiera ono informacji. Dla zmiennej losowej o dwóch równie prawdopodobnych możliwych wartościach, ($[0.5, 0.5]$), entropia wynosi 1

[bit/realizację]. Możemy oceniać "jakość" danego atrybutu na podstawie ilości informacji, jaka pozostanie, tzn. będzie jeszcze zawarta w zbiorze próbek po jego testowaniu zadanym atrybutem. Niech zbiór próbek dwóch klas zawiera *p* próbek dla klasy „pozytywnej" i *n* dla „negatywnej". Wtedy zawartość informacyjna tego zbioru próbek („przed testowaniem") wynosi:

$$H(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}log_2\frac{p}{p+n} - \frac{n}{p+n}log_2\frac{n}{p+n} \quad (5\text{-}11)$$

Wybrany atrybut *A* dzieli zbiór próbek *E* na podzbiory, E_1,… , E_v , odpowiednio do przyjmowanych przez nie wartości dla *A*, gdy *A* ma *v* różnych wartości. Oczekujemy, że każdy z podzbiorów będzie zawierał już mniej informacji po testowaniu (ze względu na atrybut A) niż przedtem. W każdym podzbiorze będzie p_i próbek o klasie pozytywnej i n_i próbek o klasie negatywnej. Oczekiwana wartość pozostałej entropii to ważona suma entropii dla każdego podzbioru:

$$H_{pozostało}(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n}\cdot H(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}) \quad (5\text{-}12)$$

Tym samym zysk informacji (ZI) odpowiadający redukcji entropii po wykonaniu testu dla atrybutu wynosi:

$$ZI(A) = H(\frac{p}{p+n},\frac{n}{p+n}) - H_{pozostało}(A) \quad (5\text{-}13)$$

W podfunkcji **WybierzAtrybut** stosujemy więc zasadę wyboru atrybutu o największym **zysku informacji**:

$$arg\max_A ZI(A) \quad (5\text{-}14)$$

Przykład 3

Dla zbioru próbek z przykładu 2 zachodzi, $p = n = 6$. Entropia w korzeniu drzewa wyniesie:

$$\mathrm{H}([6/12, 6/12]) = 1 \text{ bit.}$$

Określimy zysk informacji dla atrybutów *Osoby* i *Typ* testowanych na podanym zbiorze:

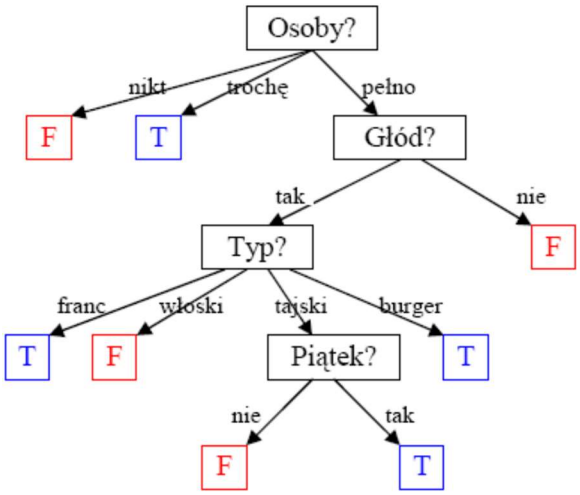
$$ZI(Osoby) = 1 - [\frac{2}{12}H([0{,}1]) + \frac{4}{12}H([1{,}0]) + \frac{6}{12}H([\frac{2}{6},\frac{4}{6}])] = 0.5409 \text{ [bit/znak]}$$

$$ZI(Typ) = 1 - [\frac{2}{12}H([\frac{1}{2},\frac{1}{2}]) + \frac{2}{12}H([\frac{1}{2},\frac{1}{2}]) + \frac{4}{12}H([\frac{2}{4},\frac{2}{4}]) + \frac{4}{12}H([\frac{2}{4},\frac{2}{4}])] = 0$$

Można sprawdzić, że atrybut *Osoby* zapewnia najwyższy zysk informacji ZI spośród atrybutów i zostanie on wybrany przez algorytm DTL do testowania próbek w korzeniu drzewa.

Przykład 4

Na Rysunek 5 pokazano drzewo decyzyjne dla problemu z przykładu 1, powstałe w wyniku uczenia przez indukcję algorytmem DTL, przy kryterium maksymalizacji zysku entropii, dla zbioru 12 podanych próbek.



Rysunek 5. Drzewo decyzyjne dla problemu z przykładu 1

3.3 Klasyfikator numeryczny

3.3.1 Definicja problemu

Klasyfikacja wektora cech (charakteryzującego obserwację obiektu czy zdarzenia) polega na podjęciu decyzji zgodnie z **funkcją decyzyjną** dla danego klasyfikatora i wiedzy nabytej uprzednio na podstawie zbioru uczącego (w procesie **uczenia**). Klasyfikator **numeryczny** polega na zastosowaniu funkcji decyzyjnej, która przypisuje liczbowy *d*-wymiarowy wektor cech, $\mathbf{c} \in \Re^d$, do dyskretnej klasy κ ze zbioru *K* klas:

$$\zeta(\mathbf{c}) = \begin{cases} \Re^d \rightarrow \{1,2,\cdots,K\} \\ \mathbf{c} \rightarrow \kappa \end{cases} \tag{5.15}$$

Ocena jakości i porównywanie różnych klasyfikatorów może uwzględniać szereg aspektów: wartość prawdopodobieństwa błędnej klasyfikacji, względnie koszt ryzyka, złożoność obliczeniowa funkcji decyzyjnej, zdolność do uogólnienia dla nieznanych próbek, zdolność do uczenia funkcji decyzyjnej, itd. Bardzo trudnym teoretycznym problemem jest określenie funkcji decyzyjnej, która optymalizuje wszystkie możliwe kryteria jakości klasyfikatora. Sensowne jest przyjęcie podstawowych założenia przy konstrukcji klasyfikatora:

1. jeśli 2 wektory cech należą do tej samej klasy to występuje pomiędzy nimi mała odległość,
2. jeśli wektory cech należą do różnych klas to pomiędzy nimi jest duża odległość.

Miarami odległości mogą być: funkcje gęstości prawdopodobieństwa, odległość *Euklidesa*, metryka Manhattan, odległość *Mahalanobisa*, itd.

3.3.2 Optymalny klasyfikator

Klasyfikator stochastyczny tworzony jest w następujących krokach:

1. zgromadź założenia (wiedzę) o analizowanej dziedzinie;
2. wyznacz ryzyko błędnej klasyfikacji (lub decyzji),
3. wyznacz funkcję decyzyjną, która minimalizuje podane ryzyko.

Każda decyzja generuje pewne koszty (ryzyko). Po wielu decyzjach możemy określić średni koszt lub ryzyko. Obliczenie średnich wartości ryzyka wymaga istnienia pełnej informacji statystycznej (w postaci zadanego rozkładu zmiennej losowej) lub uprzedniej obserwacji dającej reprezentatywny zbiór próbek.

Założenia o analizowanej dziedzinie

Statystyczne własności klas reprezentowane są przez zbiór n-wymiarowych rozkładów prawdopodobieństwa cech pod warunkiem K klas: $\{ p(\boldsymbol{c} \mid \Omega_k) \mid k=1,\ldots,\mathrm{K} \}$. Rozkład ten może mieć zadaną postać analityczną (jest to wtedy parametryczny rozkład) lub mieć charakter histogramu (rozkład nieparametryczny). W procesie *uczenia* szacowane są parametry rozkładów względnie aproksymowane są histogramami rozkłady prawdopodobieństwa w przestrzeni cech. Znany jest też rozkład prawdopodobieństwa *apriori* klas $(p(\Omega_k), k=1,\ldots,\mathrm{K})$.

Obliczenie ryzyka V

Oznaczamy koszty $\{r_{jk} = r(j \mid k) \mid j, k = 1,2, \ldots ; K\}$ błędnej klasyfikacji, w wyniku której obiekt klasy k jest błędnie klasyfikowany jako należący do klasy j. Zakładamy, że koszty te są określane przez eksperta w zależności od zastosowania. Należy też oszacować prawdopodobieństwa błędnych decyzji, $\{ p(j \mid k) \mid j, k=1,\ldots,\mathrm{K}\}$. Teraz można już obliczyć sumaryczne ryzyko (koszt) według wzoru:

$$\mathrm{V} = \sum_k \sum_j p_k \, p(j \mid k) \, r_{\mathrm{jk}} \,. \tag{5.16}$$

Reguła decyzyjna

Celem reguły decyzyjnej jest minimalizacja kosztu (ryzyka) V. Dla wyznaczenia ogólnej reguły należy przyjąć pewne założenia odnośnie funkcji kosztu. Np. niech funkcja kosztu ma binarny charakter:

$$r_{kk} = 0, \qquad r_{ik} = 1, \text{ dla } i \neq k \ (i, k = 1, \ldots, K).$$

Taka postać funkcji kosztu pozostawia główną rolę prawdopodobieństwu błędnej klasyfikacji. Analogicznie, zamiast minimalizować prawdopodobieństwo błędnej klasyfikacji, możemy zdefiniować regułę decyzyjną, która maksymalizuje prawdopodobieństwo poprawnej klasyfikacji dla każdej obserwacji. W naturalny sposób wyraża to prawdopodobieństwo warunkowe a posteriori $p(\Omega_i \mid \boldsymbol{c})$. Czyli reguła decyzyjna optymalnego klasyfikatora stochastycznego obserwacji $\boldsymbol{c}$ przyporządkowuje klasę Ω_i o największej wartości

$$p(\Omega_i \mid \boldsymbol{c}) = \mathrm{p}(\boldsymbol{c} \mid \Omega_i) \, \mathrm{p}(\Omega_i), \, , \, i = 1; \ldots, K. \tag{5.17}$$

Taką regułę decyzyjną stosuje *klasyfikator Bayesa*. Dla binarnej funkcji kosztu jest to optymalny klasyfikator w sensie minimalizacji V. Jednak zastosowanie klasyfikatora Bayesa nie zawsze jest możliwe, gdyż wymaga on pełnej informacji statystycznej o analizowanej dziedzinie.

3.3.3 Podstawowe rodzaje klasyfikatorów numerycznych

Klasyfikator według funkcji gęstości (potencjału) klas w przestrzeni cech

Należy określić postać rodziny parametrycznych funkcji (np. liniowa funkcja, wielomian 2-go stopnia) właściwej dla charakteru zbioru próbek (np. liniowa funkcja dla liniowo separowalnych danych różnych klas). Następnie w procesie uczenia klasyfikatora należy związać z każdą klasą jej konkretną funkcję wyrażającą jej znaczenie (potencjał) w przestrzeni cech. Podczas aktywnej klasyfikacji próbkom obserwacji przypisywane są klasy o maksymalnej wartości potencjału dla danego punktu przestrzeni cech. Klasyfikator o liniowych funkcjach potencjału zwykle optymalizuje kryterium złożoności obliczeniowej dla klasyfikatorów.

Klasyfikator Bayesa (klasyfikator stochastyczny)

Zakładamy, że rozkłady prawdopodobieństwa klas nad przestrzenią cech mają znaną postać (np. rozkład normalny) albo mogą być odcinkami przybliżane przez histogramy. Zadaniem procesu uczenia jest określenie parametrów rozkładu lub aproksymacja rozkładu dla każdej z rozpatrywanych klas. Zakłada się, że próbki uczące mają charakter reprezentatywny dla całości rozkładu. Wtedy klasyfikator Bayesa jest optymalny z punktu widzenia minimalizacji błędu klasyfikacji. Jako przypadki szczególne klasyfikatora Bayesa rozpatruje się klasyfikator „największej wiarygodności" i klasyfikator „geometryczny minimalnej odległości".

Klasyfikator SVM - maszyna wektorów wspierających (klasyfikator dyskryminacyjny)

Zakłada się w nim, że próbki uczące mają skończony charakter, tzn. w ogólności nie są reprezentatywne dla całości rozkładu. Podstawowy klasyfikator SVM ma charakter dyskryminacyjny dla dwóch klas – w procesie uczenia wyznaczana jest hiperpłaszczyzna oddzielająca obszary dwóch klas. Klasyfikacja *k* klas jest możliwa dzięki wielokrotnej klasyfikacji binarnej. Klasyfikator SVM jest optymalny z punktu widzenia kryterium zdolności do uogólniania dla nieistniejących próbek.

Klasyfikator neuronowy - wielowarstwowy perceptron (MLP), sieć jednokierunkowa w pełni połączona

Stosujemy sieć neuronową do aproksymacji funkcji decyzyjnych w sytuacji, gdy nie posiadany dobrze dla problemu dobranych jawnych postaci takich funkcji. Klasyfikator neuronowy dysponuje zdolnością do uczenia się funkcji decyzyjnej.

3.4 Klasyfikator według funkcji potencjału

W tym podejściu zakładamy, że:

- dla każdej klasy istnieje parametryczna funkcja (tzw. funkcja potencjału) zdefiniowana nad przestrzenią cech, charakteryzująca stopień przynależności danego punktu przestrzeni do zadanej klasy;
- wszystkie funkcje potencjału należą do jednej zadanej rodziny parametrycznych funkcji.

Najprostszą postacią takiej funkcji jest liniowa funkcja parametrów (wektor parametrów $\boldsymbol{a}$):

$$d(\boldsymbol{c}, \boldsymbol{a}) = \{\boldsymbol{a}^T \phi(\boldsymbol{c}) | \boldsymbol{a} \in \Re^{n+1} \ \}, \tag{5.18}$$

Najprostszą funkcją $\phi(\boldsymbol{c})$ jest liniowa zależność od wektora cech:

$$\phi(\boldsymbol{c}) = (1, c_1, c_2, \cdots, c_n)^T, \tag{5.19}$$

gdzie $\boldsymbol{a}$ jest wektorem o (*n*+1) elementach, a $\boldsymbol{c}$ jest wektorem cech w przestrzeni $\Re^n$.

Bardziej złożonym odwzorowaniem wektora cech będzie funkcja kwadratowa (wielomian rzędu 2) o *n* zmiennych :

$$\phi(\boldsymbol{c}) = (1, c_1, c_2, \cdots, c_n, c_1c_1, c_2c_1, \cdots, c_nc_n)^T \tag{5.20}$$

gdzie $\boldsymbol{a}$ będzie wtedy wektorem o (1+*n* + *n*(*n*+1)/2) elementach.

3.4.1 Liniowe funkcje potencjału

W binarnym problemie klasyfikacji występują tylko dwie klasy. Definiujemy dla każdej z nich liniowe funkcje potencjału, $d_1(\boldsymbol{c}, \boldsymbol{a}^{(1)})$ i $d_2(\boldsymbol{c}, \boldsymbol{a}^{(2)})$, należące do tej samej rodziny funkcji o parametrach, $\boldsymbol{a} = [a_0, a_1, ..., a_n]$:

$$d_1(\boldsymbol{c}, \boldsymbol{a}^{(1)}) = a_0^{(1)} + \sum_{i=1}^{n} a_i^{(1)} \cdot c_i, \qquad d_2(\boldsymbol{c}, \boldsymbol{a}^{(2)}) = a_0^{(2)} + \sum_{i=1}^{n} a_i^{(2)} \cdot c_i \tag{5.21}$$

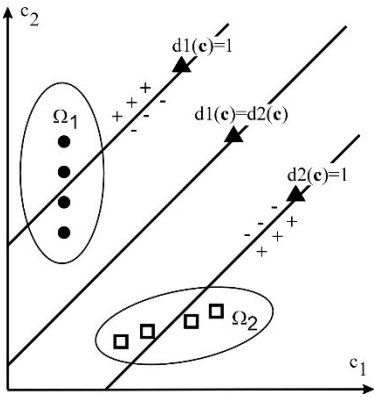
Funkcja decyzyjna ma postać:

$$d(\boldsymbol{c}) = d_1(\boldsymbol{c}, \boldsymbol{a}^{(1)}) - d_2(\boldsymbol{c}, \boldsymbol{a}^{(2)}), \tag{5.22}$$

Reguła decyzyjna: wybierz klasę Ω_1, gdy $d(\boldsymbol{c}) \geq 0$, tzn. gdy $d_1(\boldsymbol{c}, \boldsymbol{a}^{(1)}) \geq d_2(\boldsymbol{c}, \boldsymbol{a}^{(2)})$, lub wybierz klasę Ω_2 w przeciwnym razie. Możliwa jest też wersja , w której zwracany jest „brak decyzji" wtedy, gdy $d(\boldsymbol{c}) = 0$.

Przykład

Binarna klasyfikacja dla 2-wymiarowych cech - funkcja decyzyjna niejawnie wyznacza prostą, dla której punktów zachodzi równość potencjałów obu klas (rys. 13.6). Ta linia prosta separuje 2 obszary klas w przestrzeni cech o współrzędnych [c_1, c_2]. Inne linie proste, równoległe do prostej separującej obszary klas, odpowiadają kolejnym wartościom potencjału danej klasy – potencjał każdej klasy rośnie w miarę oddalania się od prostej separującej – klasy 1 przy oddalaniu się w lewo-górnym kierunku a klasy 2 w dolno-prawym kierunku.



Rysunek 6. Przykład wartości funkcji potencjału dla dwóch klas i 2-wymiarowego wektora cech.

Reguła decyzyjna dla *k* klas stanowi uogólnienie reguły dla 2 klas i wynosi:

$$\varsigma(\boldsymbol{c}) = \underset{k}{argmax} d_k(\boldsymbol{c}, \boldsymbol{a}^{(k)}) = \underset{k}{argmax}(a_0^{(k)} + \sum_{i=1}^{n} a_i^{(k)} \cdot c_i) \tag{5.23}$$

3.4.2 Uczenie się parametrów klasyfikatora

Wyznaczanie parametrów funkcji potencjału ma postać procesu aproksymacji funkcji metodą regresji. Zostanie on przedstawiony w rozdziale 4.

3.5 Klasyfikator stochastyczny Bayesa

3.5.1 Definicja

Wymagany jest model stochastyczny dla klas w przestrzeni cech w postaci:

- rozkładu apriori prawdopodobieństwa klas, $p(\boldsymbol{\Omega}) = [P(\Omega_1), P(\Omega_2), \ldots, P(\Omega_K)]$
- gęstości apriori prawdopodobieństwa warunkowego { $p(\boldsymbol{c} \mid \Omega_\kappa)$, dla wszystkich $\Omega_\kappa \in \boldsymbol{\Omega}$ }.

Zgodnie ze wzorem Bayesa prawdopodobieństwo warunkowe a posteriori wynosi:

$$p(\Omega_k|c) = \frac{p(\Omega_k)p(c|\Omega_k)}{p(c)} = \frac{p(\Omega_k)p(c|\Omega_k)}{\sum_{j=1}^{K} p(\Omega_j)p(c|\Omega_j)} \tag{5.24}$$

Reguła decyzyjna

Klasyfikator Bayesa poszukuje maksymalnego prawdopodobieństwa *a posteriori* klasy:

$$\varsigma(c) = \arg\max_{\lambda} p(\Omega_\lambda \mid c) = \arg\max_{\lambda} p(\Omega_\lambda)p(c \mid \Omega_\lambda). \tag{5.25}$$

Reguła ML

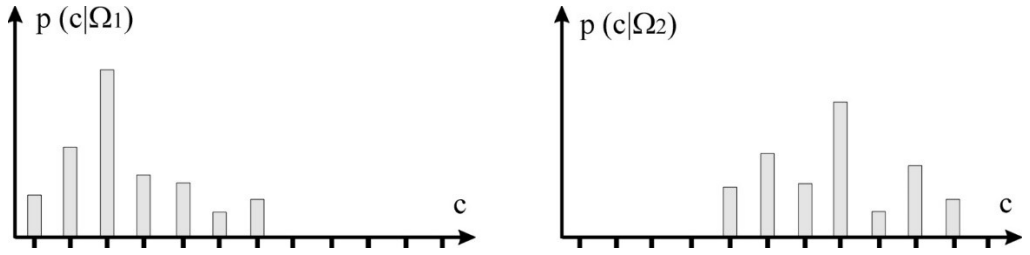
Dla jednorodnego rozkładu klas ($p(\Omega_\kappa) = p(\Omega_\lambda)$ dla wszystkich par klas) reguła decyzyjna Bayesa (5.25) sprowadza się do reguły decyzyjnej największej wiarygodności (ang. *maximum likelihood,* **ML**):

$$\varsigma(c) = \operatorname{argmax}_{\lambda} p(\Omega_\lambda|c) = \operatorname{argmax}_{\lambda} p(\Omega_\lambda)p(c|\Omega_\lambda) = \operatorname{argmax}_{\lambda} p(c|\Omega_\lambda). \tag{5.26}$$

3.5.2 Uczenie się rozkładów prawdopodobieństwa

W procesie uczenia klasyfikatora Bayesa należy wyznaczyć rozkłady prawdopodobieństwa a priori. A priori prawdopodobieństwo klas, $P(\Omega_k)$, $1 \leq k \leq K$, można wyliczyć na podstawie relatywnej częstości klas w zbiorze próbek.

Modele funkcji gęstości prawdopodobieństwa $p(\boldsymbol{c} \mid \Omega_k)$ dzielimy na parametryczne i nieparametryczne. Nieparametryczne modele polegają na zastosowaniu dyskretnego rozkładu prawdopodobieństwa w jawnej postaci (np. histogramu) (Rysunek 7). Parametryczne modele zakładają istnienie odpowiedniej rodziny funkcji gęstości prawdopodobieństwa i wymagają oszacowania ich parametrów na podstawie próbek uczących.



Rysunek 7. Histogramy cech jako nieparametryczne modelowanie gęstości prawdopodobieństwa.

Często staramy się dopasować dla *n*–wymiarowego wektora cech rozkład normalny gęstości prawdopodobieństwa, czyli *n*-wymiarową funkcję Gaussa:

$$p(\boldsymbol{c}|\Omega_k) = \frac{1}{\sqrt{|\,det(2\pi\sum_k)}} \cdot exp\left(-\frac{(\boldsymbol{c}-\mu_k)^T \cdot \sum_k^{-1} \cdot (\boldsymbol{c}-\mu_k)}{2}\right) \tag{5.27}$$

Wartości parametrów μ_k i Σ_k (dla każdej klasy Ω_k) mogą być oszacowane zgodnie z zasadą estymatora największej wiarygodności ML (ang. *maximum likelihood*). Niech $C^{(k)}$ oznacza zbiór próbek uczących (wektorów cech) dla wzorców należących do klasy Ω_k, a $\boldsymbol{\theta}^{(k)}$ będzie zbiorem parametrów rozkładu prawdopodobieństwa $p(\boldsymbol{c} \mid \Omega_k)$ (dla rozkładu normalnego, $\boldsymbol{\theta}^{(k)}=\{\mu_k, \Sigma_k\}$). Estymatorem największej wiarygodności dla klasy Ω_k jest każdy taki zestaw parametrów $\boldsymbol{\theta}^{(k)}$, który maksymalizuje wartość prawdopodobieństwa obserwowanych cech:

$$\max_{\theta^{(k)} \in \Theta} \sum_{c_j \in C^{(k)}} \log P(c_j, \boldsymbol{\theta}^{(k)}) \tag{5.28}$$

Można sprawdzić, że dla rozkładu normalnego rozwiązanie powyższego problemu (5.28) ma postać:

$$\mu_k \cong \frac{1}{N_k} \sum_{j=1}^{N_k} c_j \,; \quad c_j \in C^{(k)} \qquad \sum_k \cong \frac{1}{N_k} \sum_{j=1}^{N_k} (c_j - \mu_k)(c_j - \mu_k). \tag{5.29}$$

Dla innych postaci rozkładów niż rozkład normalny, często nie jest możliwe uzyskanie dokładnych analitycznych postaci dla estymatorów największej wiarygodności. Wtedy należy zastosować poszukiwanie maksimum funkcji ML, czyli rozwiązać „równania wiarygodności" o postaci:

$$\frac{\partial}{\partial \theta_i} \quad \sum_{c_j \in C^{(k)}} \log P(c_j, \boldsymbol{\theta}^{(k)}) = 0, \;\; \theta_i \in \boldsymbol{\theta}^{(k)} \tag{5.30}$$

W tym celu można zastosować metodę Newtona lub inną gradientową metodę iteracyjną. Przy takim podejściu wyznaczany jest w każdej iteracji optymalny kierunek w przestrzeni parametrów dla poszukiwania lokalnego maksimum „funkcji wiarygodności".

3.6 Klasyfikator geometryczny według minimalnej odległości

Klasyfikator geometryczny jest to uproszczona wersja klasyfikatora Bayesa przy zastosowaniu uproszczeń rozkładów a priori i wyrażeniu miary odległości prawdopodobieństwa przez odległość Euklidesa w przestrzeni cech.

Reguła decyzyjna klasyfikatora według minimalnej odległości

Zamiast klasy o maksymalnym a posteriori prawdopodobieństwie $p(\Omega_k \mid \mathrm{c})$ wybierz klasę o minimalnej odległości:

$$\mathrm{d}(\Omega_k, \mathrm{c}) = - \log p(\Omega_k \mid \mathrm{c}).$$

Prześledzimy wyprowadzenie powyższej reguły z reguły klasyfikatora Bayesa dla przypadku 2 klas.

Dla każdej pary klas reguła decyzyjna Bayesa – „wybierz klasę Ω_1, gdy

$$p(\Omega_1)\, p(\mathrm{c} \mid \Omega_1) > p(\Omega_2)\, p(\mathrm{c} \mid \Omega_2) \tag{5.31}$$

dla postaci rozkładu Gaussa po normalizacji obu rozkładów, tzn. po przyjęciu:

$$\sum_1 = \sum_2 \tag{5.32}$$

i po jej logarytmowaniu, odpowiada regule decyzyjnej według odległości w przestrzeni:

$$\log p(\Omega_1) + \mu_1^T \sum^{-1} c - \frac{1}{2} \mu_1^T \sum^{-1} \mu_1 > \log p(\Omega_2) + \mu_2^T \sum^{-1} c - \frac{1}{2} \mu_2^T \sum^{-1} \mu_2 \tag{5.33}$$

Specjalna postać powyższej reguły decyzyjnej otrzymana przy założeniu $p(\Omega_1) = p(\Omega_2)$, to reguła decyzyjna:

$$\mu_1^T \sum^{-1} c - \frac{1}{2} \mu_1^T \sum^{-1} \mu_1 > \mu_2^T \sum^{-1} c - \frac{1}{2} \mu_2^T \sum^{-1} \mu_2$$
$$(c - \mu_1)^T \sum^{-1} (c - \mu_1) < (c - \mu_2)^T \sum^{-1} (c - \mu_2) \tag{5.34}$$

Po kolejnym uproszczeniu rozkładów a priori – normalizacji przestrzeni cech tak, aby zachodziło $\sum$ = **1** – otrzymujemy ostatecznie:

$$(c - \mu_1)^T (c - \mu_1) < (c - \mu_2)^T (c - \mu_2) \quad \text{czyli} \quad |\, c - \mu_1 \,|^2 < |\, c - \mu_2 \,|^2 \tag{5.35}$$

Reguła decyzyjna (5.35) porównuje miary odległości Euklidesowej aktualnego wektora cech od środków rozkładów klas w przestrzeni cech i działa zgodnie z zasadą wyboru najmniejszej odległości.

3.6.1 Klasyfikator „według k sąsiadów”

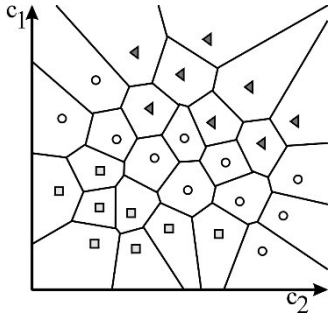
Zakładamy istnienie zbioru próbek uczących i wcześniej klasyfikowanych wektorów cech - $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, z których każda próbka względnie wektor cech należy do klasy zgodnej z regułą decyzyjną $\zeta(c_i)$ tego klasyfikatora. Każda próbka względnie każdy kolejny wektor cech staje się reprezentantem swojej klasy.

Reguła decyzyjna klasyfikatora według k sąsiadów.

Przyporządkuj nowemu wektorowi cech $\boldsymbol{c}$ tę klasę, do której należy jego najbliższy sąsiad (NN-klasyfikator, *k*=1), względnie większość spośród *k* najbliższych sąsiadów (*k*NN-klasyfikator, *k*>1).

$$\varsigma(\mathbf{c}) = \arg\min_{\varsigma(\mathbf{c}_i)} |\{|\,\mathbf{c} - \mathbf{c}_i\,|, i = 1,2,\ldots,n\} \tag{5.36}$$

W praktyce stosuje się: *k* = 3-7. Zbiór punktów przestrzeni cech, dla których najbliższym sąsiadem spośród istniejących cech jest c_i nazywany jest komórką Voronoia dla c_i. Tym samym aktualny zbiór C wyznacza podział Voronoia całej przestrzeni cech (Rysunek 8).



Rysunek 8. Ilustracja zbioru Voronoia dla 2-wymarowych wektorów cech.

Zauważmy, że powyższy podział przestrzeni cech można utożsamić z szacowaniem a posteriori prawdopodobieństw $p(\Omega_k \mid c)$ i tym samym można oszacować błąd klasyfikacji w terminach błędu dla optymalnego klasyfikatora Bayesa. Prawdopodobieństwo błędnej klasyfikacji dla klasyfikatora według *k* najbliższych sąsiadów (oznaczmy je jako P_{NN}) zbiega do prawdopodobieństwa błędu Bayesa (oznaczmy je jako P_B) wraz ze wzrostem liczby próbek uczących *N* i liczby badanych sąsiadów *k*. Prawdopodobieństwo błędnej klasyfikacji dla *kNN*-klasyfikatora (przy *K* klasach) można bowiem oszacować jako:

$$P_B \le P_{kNN} \le P_B\left(2 - P_B \frac{k}{k-1}\right) \tag{5.37}$$

czyli dla dostatecznie małych P_B w przybliżeniu zachodzi: $P_B \le P_{kNN} \le 2\,P_B$.

3.7 Klasyfikator SVM

W klasyfikatorze statystycznym zakłada się, że dysponujemy reprezentatywną próbką uczącą dla wszystkich klas. W *maszynie wektorów wspierających (nośnych)* SVM (ang. „*Support Vector Machine*”) przyjmuje się skończony charakter próbek uczących i sprowadza problem do wielokrotnej decyzji pomiędzy dwiema klasami (lub grupami klas) (oznaczanymi zwykle jako „+1”, „-1”).

Podczas procesu uczenia maszyny poszukuje się optymalnej hiperpłaszczyzny (dla przypadku liniowego) lub hiperpowierzchni (dla „nieliniowej” maszyny) separującej obszary cech dla różnych

klas. Kryteria wpływające na proces uczenia to minimalizacja tzw. błędu empirycznego i rozmiar Vapnika-Czervonenkisa.

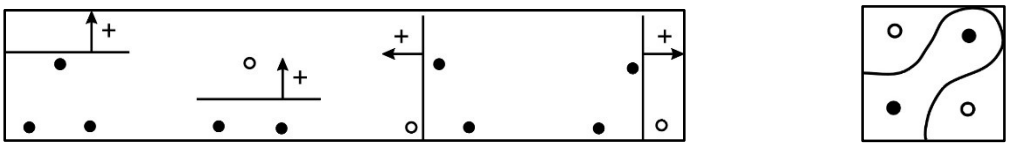
Rozmiar Vapnika-Czervonenkisa (rozmiar VC) *h* jest miarą dla zbioru funkcji rozdzielających. Dla problemu dwóch klas *h* oznacza maksymalną liczbę punktów w przestrzeni cech, które mogą zostać rozdzielone we wszystkie możliwe sposoby – liczba takich podziałów wynosi 2^h. Wystarczy przy tym, aby dla każdego sposobu podziału istniał przynajmniej jeden zbiór punktów o mocy *h*, tzn. istnieje wtedy przynajmniej jedna prawidłowa funkcja rozdzielająca. Specyficznym zbiorem funkcji rozdzielających jest zbiór zorientowanych hiperpłaszczyzn:

$$d_{\tilde{a}}(\mathbf{c}) = \mathbf{c}^T \boldsymbol{a} + a_0, \quad \tilde{a} = \begin{pmatrix} a_0 \\ \boldsymbol{a} \end{pmatrix} \tag{5.38}$$

Za ich pomocą określamy to, czy wektor cech $\mathbf{c}$ leży po „dodatniej" stronie lub po „ujemnej" stronie płaszczyzny czy też na samej płaszczyźnie rozdzielającej. Można pokazać, że rozmiar Vapnika-Chervonenkisa zbioru zorientowanych hiperpłaszczyzn w *n*-wymiarowej przestrzeni cech $\Re^n$ wynosi: $h = n + 1$.

Przykład.

Na płaszczyźnie zbiór trzech punktów daje się rozdzielić za pomocą zorientowanej prostej na, $2^3 = 8$, sposobów (Rysunek 9). Ta własność nie zachodzi już dla czterech punktów, czyli dla płaszczyzny (n=2, h=3).



(a) (b)

Rysunek 9. Ilustracja rozdziału zbioru punktów na płaszczyźnie: (a) 4 podziały uzyskane dzięki jednej zorientowanej prostej, pozostałe 4 powstają po zmianie zorientowania prostej; (b) podział 4 punktów jedną prostą w ogólności nie jest możliwy.

3.7.1 Podstawowy SVM

Zakładamy istnienie *N* próbek dwóch klas i to, że obszary dwóch klas są liniowo separowalne. Próbki są etykietowane jako „dodatnie próbki" (etykieta $y_i = 1$) lub „negatywne próbki" ($y_i = -1$). Klasyfikator SVM wyznacza hiperpłaszczyznę o parametrach $\tilde{\boldsymbol{a}} = [a_0, a_1, \cdots, a_n]^T$, która rozdziela próbki uczące $\{ {}^j\boldsymbol{c} \mid j=1,2,\ldots,N\}$ i zdefiniowana jest następująco:

$$d^*_{\tilde{a}}(\mathbf{c}): \ \mathbf{c}^T \boldsymbol{a} + a_0 = 0; \qquad \tilde{a} = \begin{pmatrix} a_0 \\ \boldsymbol{a} \end{pmatrix} \tag{5.39}$$

Przy tym spełnione są ograniczenia

$${}^j\mathbf{c}^T \boldsymbol{a} + a_0 \geq +1, \quad if \quad y_j = +1\,; \qquad {}^j\mathbf{c}^T \boldsymbol{a} + a_0 \leq -1, \quad if \quad y_j = -1$$

czyli zapisane w zwarty sposób

$$y_j({}^j\mathbf{c}^T \boldsymbol{a} + a_0) \geq +1, \quad \forall\, {}^j\mathbf{c} \in \omega \tag{5.40}$$

Dla hiperpłaszczyzny (5.39) zachodzą zależności:

$$\mathbf{n} = \frac{-\boldsymbol{a}}{\sqrt{\boldsymbol{a}^T \boldsymbol{a}}} = \frac{-\boldsymbol{a}}{|\boldsymbol{a}|}, \qquad \text{Jednostkowy wektor normalny}$$

$$s_0 = \frac{-a_0}{|\boldsymbol{a}|}, \qquad \text{Odległość od początku układu} \tag{5.41}$$

$$s_{\mathbf{c}} = \frac{-(\boldsymbol{a}^T \boldsymbol{c} + a_0)}{|\boldsymbol{a}|},$$ Odległość od punktu *c*

„Pozytywna" odległość punktu od hiperpłaszczyzny zachodzi wtedy, gdy punkt leży w kierunku wektora normalnego do powierzchni. Dwie hiperpłaszczyzny „wspierające" są położone w „znormalizowanej" odległości od hiperpłaszczyzny (5.39) – dla nich zachodzi:

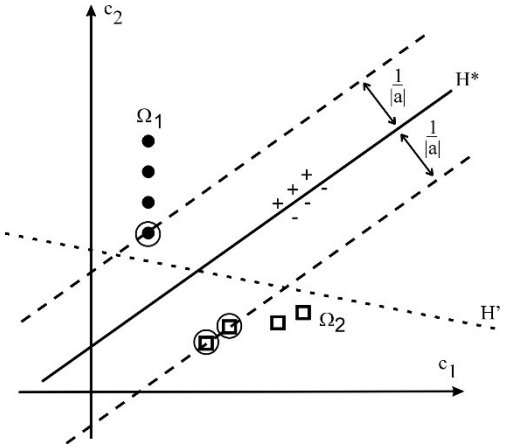
$$d_1(\boldsymbol{c})\colon \boldsymbol{c}^T\boldsymbol{a} + a_o = 1 \qquad \text{lub} \qquad d_{-1}(\boldsymbol{c})\colon \boldsymbol{c}^T\boldsymbol{a} + a_o = -1\,. \tag{5.42}$$

Obie hiperpłaszczyzny posiadają ten sam wektor normalny $\boldsymbol{a}$, tzn. są „równoległe" do siebie. Ich odległości od środka układu wynoszą odpowiednio: $|1 - a_o|/|\boldsymbol{a}|$ i $|-1 - a_o|/|\boldsymbol{a}|$, czyli ich wzajemny odstęp wynosi $(2/|\boldsymbol{a}|)$. Pomiędzy tymi dwiema hiperpłaszczyznami nie mogą leżeć żadne próbki uczące.

W procesie uczenia stosowane jest kryterium maksymalizacji odstępu obu „wspierających" hiperpłaszczyzn, czyli minimalizacja $|\boldsymbol{a}|^2$ przy jednoczesnym spełnianiu przez próbki uczące ograniczeń (5.40).

Przykład.

Dana jest hiperpłaszczyzna H*, optymalna w sensie klasyfikatora SVM (Rysunek 10). Te z próbek uczących, które leżą dokładnie na hiperpłaszczyznach związanych z H*, tzn. dla których zachodzi równość w równaniach (13.23) nazywamy „wektorami wspierającymi". Są one wystarczające i konieczne do tego, aby wyznaczyć hiperpłaszczyznę optymalną – pozostałe próbki uczące mogą być pominięte bez szkody dla wyznaczenia H*. Z drugiej strony pominięcie jakiegoś wektora wspierającego zmieniłoby rozwiązanie dla H*.



Rysunek 10. Ilustracja hiperpłaszczyzny rozdzielającej H klasyfikatora SVM. Mogą istnieć inne płaszczyzny rozdzielające (np. H') ale nie są one optymalne w sensie przyjętych kryteriów.*

Zadanie uczenia podstawowego klasyfikatora SVM

Celem jest wyznaczenie hiperpłaszczyzny: $d_{\tilde{\boldsymbol{a}}}(\mathbf{c})\colon \mathbf{c}^T\boldsymbol{a} + a_0 = 0$; dzięki minimalizacji funkcji celu

$$\min_{\tilde{a}\in R^n} f(\tilde{\boldsymbol{a}}) = \min_{\tilde{a}\in R^n}\left(\frac{1}{2}|\tilde{\boldsymbol{a}}|^2\right) \tag{5.43}$$

przy spełnieniu *N* warunków dodatkowych, $C_i(\boldsymbol{a}) \geq 0$ (i =1, ..., N) o postaci:

$$y_j({}^j\mathbf{c}^T\boldsymbol{a} + a_0) - 1 \geq 0, \quad \forall\, {}^j\mathbf{c} \in \Omega \tag{5.44}$$

Reguła decyzyjna jest postaci:

$$\zeta(c) = \begin{cases} \Omega_1, & gdy \quad d_{\tilde{\boldsymbol{a}}}(\boldsymbol{c}) \geq 0 \\ \Omega_2, & gdy \quad d_{\tilde{\boldsymbol{a}}}(\boldsymbol{c}) < 0 \end{cases} \tag{5.45}$$

Problem (5.43-5.44) znany jest w teorii optymalizacji jako **wypukłe programowanie kwadratowe** z nierównościowymi ograniczeniami liniowymi.

3.7.2 Wektory wspierające

Okazuje się, że równanie szukanej hiperpłaszczyzny możemy wyrazić alternatywnie do parametrów $\tilde{\boldsymbol{a}}$ za pomocą próbek uczących stanowiących „wektory wspierające" dla tej hiperpłaszczyzny:

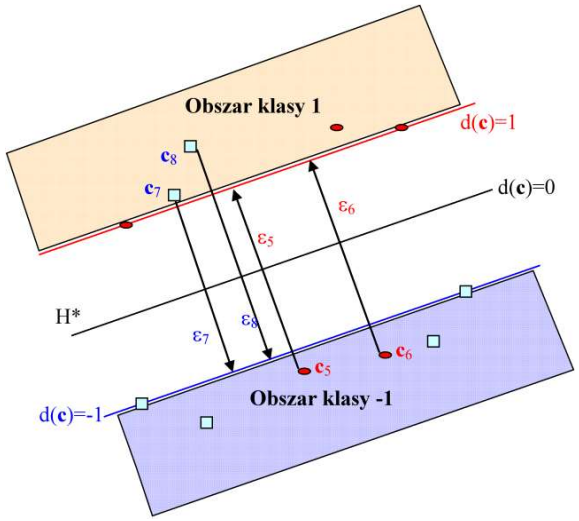
$$H^*: d_{\tilde{\boldsymbol{a}}}(\mathbf{c}) = \sum_{j=1}^{N} \vartheta_j y_j (\mathbf{c}^T \, ^j\mathbf{c}) + a_0 = 0 \tag{5.46}$$

gdzie wagi ϑ_j poszczególnych iloczynów skalarnych wektorów próbek są nieujemnymi mnożnikami Lagrange'a. Równanie (5.46) wyraża fakt, że do wyznaczenia hiperpłaszczyzny wystarczą produkty skalarne wybranych wektorów cech. Ta obserwacja pozwoli nam na łatwe uogólnienie liniowego SVM do nieliniowego SVM.

Dla próbek (wektorów cech) leżących dokładnie na hiperpłaszczyźnie mnożniki wynoszą: ϑ_j=0 lub ϑ_j>0, a dla próbek nie leżących na hiperpłaszczyźnie mnożniki są zerowane: ϑ_j = 0. Do obliczenia optymalnej hiperpłaszczyzny wchodzą tylko wektory cech o aktywnych warunkach dodatkowych (ϑ_j>0) – są to **wektory wspierające** - wszystkie pozostałe próbki uczące są zbędne.

3.7.3 SVM „z szumem"

Jeśli zbiór próbek nie jest w pełni liniowo separowalny, ale błędnie klasyfikowane próbki zdarzają się w miarę rzadko, to możemy je traktować jako zaszumione dane. Uwzględniamy je w funkcji celu liniowej SVM „z szumem" () w postaci dodatkowej składowej „kary":



Rysunek 11. Dodajemy element "kary" do funkcji celu dla każdej błędnie zaklasyfikowanej próbki uczącej. W powyższym przypadku proporcjonalnie do odległości ε_5 i ε_6 (dla próbek klasy 1) i ε_7 i ε_8 (dla próbek klasy -1).

$$\min_{\tilde{a} \in R^n} f(\tilde{\boldsymbol{a}}) = \min_{\tilde{a} \in R^n} (\frac{1}{2} |\tilde{\boldsymbol{a}}|^2 + C \sum_{j=1}^{N} \varepsilon_j) \tag{5.47}$$

gdzie C jest ustalonym parametrem, ważącym wpływ szumu na łączną funkcję celu, a wartości ε_j to odległości błędnych próbek od hiperpłaszczyzn wspierających dla ich właściwej klasy. Ograniczenia dodatkowe przyjmują wtedy postać:

$$y_j({}^j\mathbf{c}^T\boldsymbol{a} + a_0) \geq 1 - \varepsilon_j, \quad \forall\, {}^j\mathbf{c} \in \omega,\ \varepsilon_j \geq 0 \tag{5.48}$$

Wprawdzie zamiast (*n*+1) zmiennych mamy teraz (*n*+1+*N*) zmiennych, ale okazuje się, że problem tego zmodyfikowanego uczenia SVM jest analogiczny do uczenia podstawowego SVM (5.43-5.44) - zmienia się jedynie zakres wartości mnożników Lagrange'a i wynosi on teraz, $0 \leq \vartheta_j \leq C$, podczas gdy suma wag ze znakiem nadal pozostaje zerowa:

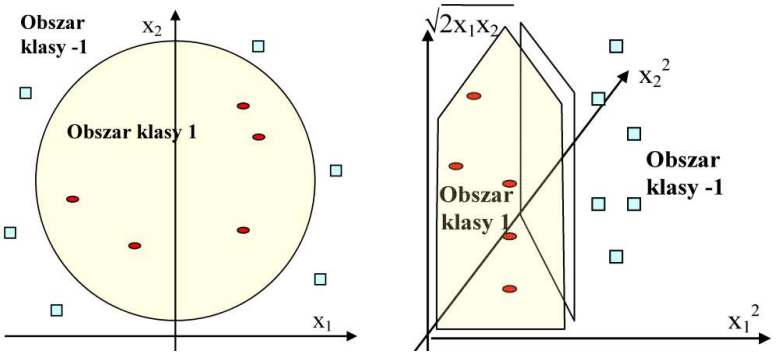
$$0 = \sum_{j=1}^{N} \vartheta_j y_j \tag{5.49}$$

3.7.4 Nieliniowe SVM

W złożonych problemach praktycznych dane uczące nie będą liniowo separowalne ani też separowalne po uwzględnieniu szumu. Aby moc w takiej sytuacji zastosować klasyfikator SVM należy dokonać odpowiedniego nieliniowego przekształcenia przestrzeni, np. przekształcając ją w wyżej wymiarową przestrzeń *F*(***c***). W tym przekształceniu zastępujemy wektory ***c*** i ${}^j\boldsymbol{c}$ podane w równaniu (5.46) przez *F*(***c***) i *F*(${}^j\boldsymbol{c}$):

$$d_{\tilde{\boldsymbol{a}}}(\mathbf{c}) = \sum_{j=1}^{N} \vartheta_j y_j (F(\mathbf{c})^T F({}^j\mathbf{c})) + a_0 = 0 \tag{5.50}$$

Często produkt skalarny wektorów, $F(\boldsymbol{c})^T F({}^j\mathbf{c})$, może być obliczony analitycznie co oznacza, że nie będzie potrzeby obliczania wartości funkcji $F(\ )$ dla każdej próbki z osobna. Np. jeśli ($F_1 = x_1^2$, $F_2 = x_2^2$, $F_3 = \sqrt{2}x_1x_2$) to okazuje się, że: $F(\mathrm{c})^T\ F({}^j\mathrm{c}) = (\mathrm{c} \cdot {}^j\mathrm{c})^2$ (Rysunek 12). Szukana (nieliniowa) funkcja decyzyjna jest postaci: $x_1^2 + x_2^2 \leq 1$.



(a) (b)

Rysunek 12. Liniowo nieseparowalny zbiór 2-wymiarowych próbek uczących (próbki klasy +1 zaznaczono jako wypełnione, a próbki klasy -1 jako okręgi). Po odwzorowaniu tych danych do przestrzeni 3-wymiarowej możliwa jest linearyzacja funkcji decyzyjnej

Takie wyrażenie nazywane jest funkcją "jądra" tego przekształcenia (ang. *kernel function*). Przykłady funkcji jądra to :

- wielomian: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{\mathrm{T}}\mathbf{y} + 1)$
- funkcja radialna: $K(\mathbf{x}, \mathbf{y}) = exp\left(-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}\right)$
- postać neuronowa: $K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\, \mathbf{x}^{\mathrm{T}}\mathbf{y} - \delta)$

4 Zadanie aproksymacji funkcji

Uczenie się aproksymacji funkcji jest kolejną, po uczeniu się pojęć i klasyfikacji, odmianą uczenia indukcyjnego. W zadaniu aproksymacji poszukujemy reprezentacji nieznanej funkcji, $f(\mathbf{x})$: $\mathbf{X} \rightarrow \Re$, o wartościach rzeczywistych. Wyznaczyć należy taką funkcję, $h(\mathbf{x}; \mathbf{p})$, z parametrami $\mathbf{p} = [p_1, p_2, \ldots, p_n]^T$, której wartości w badanej dziedzinie **X** dostatecznie mało różnią się, w sensie przyjętego kryterium, od odpowiednich wartości funkcji $f(\mathbf{x})$.

Informacja trenująca - próbka ucząca to para $(\mathbf{x}, f(\mathbf{x}))$ dla $\mathbf{x} \in \mathbf{X}$.

Dla większości algorytmów jako kryterium oceny wygodne jest stosowanie błędu średniokwadratowego zdefiniowanego jako:

$$e_p = \frac{1}{N}\sum_{x_i \in T}^{\Sigma}(f(\mathbf{x}_i) - h(\mathbf{x}_i, \mathbf{p}))^2 \qquad (5.51)$$

gdzie N oznacza liczbę próbek uczących w zbiorze treningowym $\mathbf{T}$.

Często tak aproksymowana funkcja pełni następnie rolę funkcji decyzyjnej lub funkcji potencjału w zagadnieniach klasyfikacji i charakteryzuje ona stopień przynależności danej próbki do zadanej klasy.

4.1 Metodyka uczenia się aproksymacji funkcji

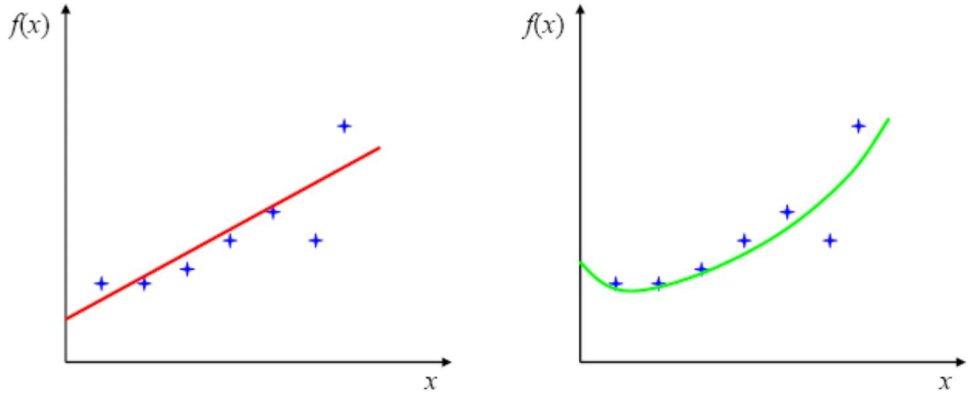
Niech f oznacza nieznaną nam docelową funkcję. Znane są **przykłady** (obserwacje) tej funkcji dla zadanego wejścia x – czyli pojedyncza obserwacja (**próbka ucząca**) to para $(x, y=f(x))$.

Zadanie uczenia aproksymacji funkcji: mając dany zbiór próbek uczących znaleźć funkcję **(hipotezę)** h, aproksymującą nieznaną funkcję, tzn. $h \approx f$.

Jak zmierzyć jakość hipotezy, tzn. bliskość hipotezy i rzeczywistej funkcji, tzn. czy $h \approx f$? Jakość hipotezy h można wyrazić poprzez ocenę jej zdolności do generalizacji, tzn. tego, jak dobrze przewiduje ona wartości funkcji f dla nieznanych dotąd obserwacji.

Przykład

Hipoteza jest **zgodna z próbkami** uczącymi, jeśli wszystkie je spełnia. Załóżmy, że znana jest aproksymacja zbioru punktów na płaszczyźnie linią prostą lub krzywą (funkcja 1-wymiarowa) (Rysunek 13). Hipoteza w postaci liniowej funkcji nie jest zgodna z wieloma próbkami. Dlatego kontynuujemy szukanie hipotezy o postaci wielomianu wyższego rzędu. Może być wiele funkcji zgodnych z próbkami uczącymi. Kierujemy się wtedy zasadą wyboru funkcji najprostszej spośród zgodnych funkcji.



Rysunek 13. Aproksymacja zbioru próbek funkcja liniową i kwadratową

4.2 Funkcja liniowa

W klasycznym zadaniu **aproksymacji parametrycznej** poszukiwana funkcja ma postać parametryczną, zdefiniowaną nad przestrzenią cech jako:

$$h(\mathbf{x}, \mathbf{p}) = \mathbf{p}^T \phi(\mathbf{x}), \quad \mathbf{p} \in \Re^m, \phi(\mathbf{x}) \in \Re^m \tag{5.52}$$

Każdy przykład (próbka ucząca), **x** ∈ **X** , jest reprezentowany przez wektor cech, **c** = **ϕ**(**x**), gdzie **ϕ** = [φ_0, ... , φ_m]^T jest wektorem funkcji wyznaczających wartości cech. Dla każdej klasy funkcja *h*() ma tę samą postać, a różni się jedynie unikalnym wektorem wartości parametrów ***p***. W istocie jest to funkcja liniowa nad przestrzenią cech, gdzie jednak tzw. **funkcje bazowe**, φ_i(x), są potencjalnie nieliniowe. **Etapy** w aproksymacji funkcji parametrycznej:

1. poszukiwanie zestawu funkcji bazowych **ϕ**(x),
2. minimalizacja błędu e_p względem wektora parametrów ***p***.

4.2.1 Regresja liniowa

Liniowa funkcja bazowa przedstawia bezpośrednią zależność wektora cech od próbki:

$$\mathbf{c} = \phi(\mathbf{x}) = [1, x_1, x_2, \cdots, x_m]^T$$

gdzie **c** jest wektorem o (*m*+1) elementach. Z kolei przykład nieliniowej funkcji bazowej to **funkcja kwadratowa** (wielomian rzędu 2) o *m* zmiennych (składowych **x**):

$$\phi(\mathbf{x}) = [1, x_1, x_2, \cdots, x_n, x_1x_1, x_2x_1, \cdots, x_nx_n]^T$$

Teraz **c** będzie wektorem (1+*m* + *m*(*m*+1)/2)-elementowym i tyle też potrzebnych będzie parametrów w wektorze ***p***.

Nieliniowe funkcje bazowe wyższego rzędu w ogólności również mogą zależeć od zbioru parametrów, tak jak poszukiwana aproksymacja funkcji *f*(**x**).

Błąd średniokwadratowy $\boldsymbol{e_p}$ to funkcja kwadratowa, więc dla zmiennej *x* o jednym wymiarze i **liniowych funkcji** *f*, *h* ma ona jedno rozwiązanie i osiąga minimum w miejscu zerowania się pochodnej. Mamy więc w punkcie rozwiązania: $\frac{d\, e_p}{d\, \boldsymbol{p}} = 0$. W celu wyznaczenia aproksymacji **wielowymiarowej** funkcji *f* , poszczególne wymiary potraktujemy jako niezależne i rozwiążemy układ *m* równań o *m* niewiadomych parametrach p_i :

$$\frac{d\, e_p}{d\, p_i} = 0 \ , \ i = 0,1,\ldots,m$$

W przypadku funkcji liniowych znalezione rozwiązanie jest globalnym minimum funkcji błędu.

Przykład - wyznaczenie parametrów funkcji potencjału klasyfikatora

Załóżmy rodzinę liniowych funkcji o parametrach $\boldsymbol{p}$, $h_{\boldsymbol{p}}(\boldsymbol{x})$, gdzie $\boldsymbol{p} = [p_0, p_1, ..., p_m]$. Czyli:

$$y = h_p(\mathbf{x}) = p_0 + \sum_{i=1}^{m} p_i \cdot x_i \text{ `}$$

Przykłady trenujące nie musza leżeć na jednej prostej (płaszczyźnie, hiperpłaszczyźnie) w przestrzeni, ale próbujemy jednak aproksymować je funkcją liniową. Danych jest *N* próbek uczących. Można ułożyć układ równań o (*N*) wierszach i (*m*+1) kolumnach (dla *m*+1 parametrów w wektorze $\boldsymbol{p}$). Uzupełniamy każdą próbkę $\mathbf{x}_i$ o zerową składową ($x_{i,0} = 1$) i otrzymujemy macierz współczynników próbek $\mathbf{X_1}$:

$$\boldsymbol{X_1} = \begin{bmatrix} 1 & \boldsymbol{x}_1^T \\ 1 & \boldsymbol{x}_2^T \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,m} \\ 1 & x_{2,1} & \cdots & x_{2,m} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,m} \end{bmatrix}$$

Wektor wyników funkcji, $\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^{\boldsymbol{T}}$, dla *N* próbek uczących, a nieznany wektor błędów aproksymacji to: $\boldsymbol{e} = \mathbf{X_1}\,\boldsymbol{p} - \mathbf{y}$. Stąd średni błąd kwadratowy aproksymacji wynosi:

$$||\boldsymbol{e}||^2 = \frac{1}{N}\sum_{i=1}^{N} e_i^2 .$$

Minimum powyższej funkcji błędu znajdujemy metodą „najmniejszych kwadratów" (MNK) - przyrównujemy pochodne cząstkowe funkcji błędu względem kolejnych *m* parametrów do zera i rozwiązujemy tak wyznaczony układ *m* równań.

Dla klasyfikatora według funkcji potencjału o *K* klasach (punkt 3.4) przyjmiemy dla klasy *k*:

$y_i = h_{\boldsymbol{p}}^{(k)}(\mathbf{x}_i) = 1$, gdy $\mathbf{x}_{\boldsymbol{i}} \in \Omega_k$;

$y_i = h_{\boldsymbol{p}}^{(k)}(\mathbf{x}_i) = -1$, gdy $\mathbf{x}_{\boldsymbol{i}} \notin \Omega_k$.

Dla klasy o indeksie *k* istnieje N_k próbek uczących etykietowanych przynależnością do klasy. Dla każdej klasy można ułożyć układ równań o ($N = \sum N_k$) wierszach i (*m*+1) kolumnach (dla *m*+1 zmiennych tworzących wektor $\boldsymbol{p}^{(k)}$). Uzupełnijmy każdy wektor **x** o zerową składową ($x_0 = 1$). Wektor oczekiwanych wyników (-1 lub 1) oznaczmy przez $\mathbf{y}^{(k)}$. Ostatecznie dla każdej klasy *k* zapiszemy odrębny układ równań o postaci:

$$\mathbf{y}^{(k)} = \mathbf{X}_1\,\mathbf{p}^{(k)}, \tag{5.53}$$

i rozwiążemy go metodą najmniejszych kwadratów (MNK).

4.2.2 Optymalizacja MNK

Metoda **najmniejszych kwadratów** (MNK) jest podstawowym narzędziem optymalizacji. Wyjaśnimy jej istotę dla przypadku 2-wymiarowego. Wtedy celem metody MNK jest znalezienie liniowej zależności pomiędzy wielkościami *y* i *x*, danej za pomocą parametrów $\boldsymbol{p} = [a, b]$:

$$y = a\,x + b \ , \text{ lub } \ a\,x + b - y = 0.$$

Na podstawie *N* obserwacji punktów (x_i , y_i) mamy *N* równań :

$$\begin{aligned} a\,x_1 + b - y_1 &= e_1 \\ a\,x_2 + b - y_2 &= e_2 \\ &\cdots\cdots\cdots\cdots \end{aligned} \tag{5.54}$$

$$a\,x_N + b\; - y_N = e_N$$

przy czym wprowadziliśmy wektor błędu aproksymacji: $\boldsymbol{e} = [\, e_1\,,\, e_2\,,\, \ldots\,,\, e_N\,]^{\mathrm{T}}$.

Celem optymalizacji jest minimalizacja sumarycznego błędu kwadratowego:

$$U(a,\, b) = e_1^2 + e_2^2 + \ldots + e_{\mathrm{N}}^2 = \sum e_{\mathrm{i}}^2 = ||\, \boldsymbol{e}\, ||^2 \tag{5.55}$$

$$U(a,\, b\,) = \sum (a\, x_i + b - y_i\,)^2$$

Minimum funkcji U wystąpi w punkcie zerowania się pochodnych cząstkowych:

$$\frac{\partial U}{\partial a} = 0; \quad \frac{\partial U}{\partial b} = 0$$

$$\frac{1}{2}\frac{\partial U}{\partial x} = \sum (a\, x_i + b - y_i) \cdot x_i = 0; \qquad \frac{1}{2}\frac{\partial U}{\partial y} = \sum (a\, x_i + b - y_i) = 0 \tag{5.56}$$

Otrzymujemy układ 2 równań z 2 niewiadomymi:

$$a\;\sum x_i{}^2 + \;b\;\sum x_i \;= \sum (x_i\, y_i\,)$$

$$a\;\sum x_i + \;b\, N \;= \sum y_i$$

Jeśli istnieje rozwiązanie powyższego układu to przyjmuje ono postać:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{N\sum x_i^2 - (\sum x_i)^2} \begin{bmatrix} N & -\sum x_i \\ -\sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} \sum x_i\, y_i \\ \sum y_i \end{bmatrix} \tag{5.57}$$

4.3 Funkcja nieliniowa

Jeśli poszukiwana funkcja $h_p(x)$ **nie jest liniowa** to metoda MNK nie prowadzi bezpośrednio do układu równań liniowych. Zamiast tego stosujemy metodę iteracyjnego poprawiania oszacowania parametrów $\boldsymbol{p}(k)$ takiej nieliniowej funkcji, przesuwając się w przestrzeni parametrów wzdłuż kierunku ujemnego gradientu funkcji błędu. Jest to reguła spadku gradientu (ang. gradient descent):

$$\boldsymbol{p}(k) = \boldsymbol{p}(k-1) + \Delta\boldsymbol{p},$$

gdzie $\Delta\boldsymbol{p} = -\beta \cdot \nabla_p \boldsymbol{e_p}$, jest przeskalowanym wektorem aktualnych (w iteracji *k*) pochodnych cząstkowych:

$$\nabla_p \boldsymbol{e_p} = [\frac{\partial \boldsymbol{e_p}}{\partial p_0}, \frac{\partial \boldsymbol{e_p}}{\partial p_1}, \cdots, \frac{\partial \boldsymbol{e_p}}{\partial p_m}]$$

a współczynnik skalujący, $\beta \in (0,1]$. W *k*-tej iteracji wyznaczamy wektor modyfikacji $\Delta\boldsymbol{p}$ jako:

$$\Delta\mathbf{p} = \beta \frac{2}{N} \sum_{x_i \in T}^{\Sigma} (\boldsymbol{y_i} - h(\mathbf{x}_i, \boldsymbol{p}(k))) \cdot \nabla_p h(\boldsymbol{x}_i, \boldsymbol{p}(k)) \tag{5.58}$$

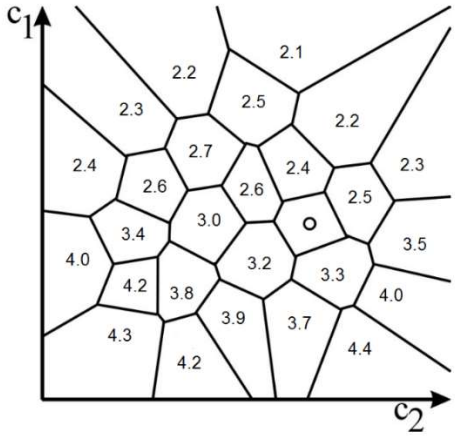
Posługiwanie się regułą spadku gradientu dla funkcji nieliniowych *h* nie daje gwarancji, że znalezione minimum dla funkcji błędu jest jednocześnie minimum globalnym tej funkcji, gdyż w warunkach występowania wielu minimów lokalnych osiągnięte tą metodą minimum zależy od przyjętej wartości początkowej parametrów $\boldsymbol{p}$.

4.4 Model pamięciowy aproksymacji funkcji

Przedstawimy teraz zupełnie odmienne podejście do uczenia się aproksymacji funkcji. Tzw. pamięciowa metoda uczenia się nie przetwarza przykładów trenujących a tylko te przykłady zapamiętuje.

Załóżmy istnienie zbioru próbek uczących, $\mathrm{T} = \{\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_{\mathrm{N}}\}$, dla których znane są wartości nieznanej funkcji $f(\mathrm{c_i})$. W pamięciowym modelu uczenia każda próbka zostaje zapamiętana, staje się reprezentantem funkcji a wartość jej etykiety stanowi aproksymację wartości funkcji dla

„zajmowanego przez nią” obszaru w przestrzeni reprezentacji. Zbiór punktów przestrzeni cech, dla których najbliższym sąsiadem spośród istniejących cech jest c_i nazywamy *komórką Voronoia* dla c_i. Zbiór przykładów uczących *T* wyznacza *podział Voronoia* całej przestrzeni cech (Rysunek 14).



Rysunek 14. Ilustracja podziału Voronoia przestrzeni 2-wymarowych cech.

Zasada aproksymacji według *najbliższego sąsiada*

Aproksymuj wartość funkcji w punkcie ***x*** przez wartość pamiętanej próbki ***c*** położonej najbliżej zadanej wartości ***x*** **:**

$$h(\mathbf{x}) = f(\boldsymbol{c}_{opt})$$

gdzie $\mathbf{c}_{opt} = arg\min_{c_i \in T} \|\mathbf{x} - \mathbf{c}_i\|$

Aproksymacja według *k sąsiadów*

Aproksymuj wartość funkcji w punkcie **x** przez średnią wartość *k* pamiętanych próbek $\mathbf{c}_i$ położonych najbliżej zadanemu punktowi **x** spośród próbek zbioru *T*:

$$h(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} f(\mathbf{c}_i)$$

5 Pytania

1. Omówić algorytm uczenia drzewa decyzyjnego DTL.
2. Na czym polega stosowanie entropii dla wyboru testu podczas uczenia DTL?
3. Omówić funkcję decyzyjną klasyfikatora według funkcji potencjału.
4. Przedstawić klasyfikator Bayesa i zasadę jego uczenia.
5. Omówić klasyfikator minimalno-odległościowy i jego związek z klasyfikatorem Bayesa.
6. Przedstawić klasyfikator kNN.
7. Na czym polega podstawowy klasyfikator SVM? Co to są wektory wspierające?
8. Na czym polega problem „programowania kwadratowego” z ograniczeniami liniowymi?
9. Jak definiujemy „zaszumiony” SVM i nieliniowy SVM?
10. Na czym polega zadanie liniowej regresji?

11. Wyjaśnić tryb uczenia liniowej funkcji potencjału dla klasyfikatora binarnego.
12. Na czym polega zadanie nieliniowej regresji?
13. Omówić model pamięciowy w uczeniu się aproksymacji funkcji.