Sieci typu LVQ i klasyfikator kNN

Dygresja - metody bazujące na podobieństwie

- Podstawowa zasada:
  - Elementy podobne powinny należeć do tej samej klasy -> inspiracja kognitywistyczna
- Problem: co to znaczy podobne i jak zdefiniować podobieństwo?
  - W.Duch „Similarity based methods a general framework for classification approximation and association.” Control and Cybernetics, 2000
  - Podobieństwo to różne miary odległości lub ich odwrotności – (miary podobieństwa)

Klasyfikator najbliższego sąsiada

Uczenie:

- Zapamiętaj położenie wszystkich wektorów zbioru treningowego

Testowanie:

- Dla każdego wektora testowego wyznacz jego odległość do wszystkich wektorów zbioru treningowego.
- Wybierz spośród wszystkich odległości wektor najbliższy (najbardziej podobny) danego wektora testowego
- Przypisz etykietę wektorowi testowemu równą etykiecie najbliższego sąsiada.

Klasyfikator najbliższego sąsiada

Klasyfikator 1NN (1 najbliższego sąsiada – ang. one nearest neighbor)

$$ii = \arg\min_i \left(D\left(\mathbf{x}, \mathbf{p}_i\right)\right)$$

$$c\left(\mathbf{x}\right) \rightarrow c\left(\mathbf{p}_{ii}\right)$$

D(**x**,**p**) – odległość pomiędzy wektorami **x** i **p**

$$ii = \arg\max_i \left(S\left(\mathbf{x}, \mathbf{p}_i\right)\right)$$

$$c\left(\mathbf{x}\right) \rightarrow c\left(\mathbf{p}_{ii}\right)$$

S(**x**,**p**) – podobieństwo pomiędzy wektorami **x** i **p**

Podobieństwo a odległość

Podobieństwo jest odwrotnością odległości:

- Metody transformacji odległości do podobieństwa i odwrotnie np.:

$$S(\mathbf{x},\mathbf{y}) = \exp\left(-D(\mathbf{x},\mathbf{y})^{\alpha}\right)$$

$$S(\mathbf{x},\mathbf{y}) = \frac{1}{D(\mathbf{x},\mathbf{y})+1}$$

Różne miary odległości

$$D_E(\mathbf{x},\mathbf{y})^2 = \sum_{i=1}^{n}(x_i - y_i)^2$$ Odległość Euklidesa

$$D_M(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}|x_i - y_i|$$ Odległość Manhattan

$$D_C(\mathbf{x},\mathbf{y}) = \max|x_i - y_i|$$ Odległość Czebyszewa

$$D_{Mi}(\mathbf{x},\mathbf{y})^{\alpha} = \sum_{i=1}^{n}|x_i - y_i|^{\alpha}$$ Odległość Minkowskiego

$$D_H(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}\begin{cases}0 \Rightarrow x_i = y_i\\ 1 \Rightarrow x_i \neq y_i\end{cases}$$ Odległość Hamminga

Odległość Heterogeniczna

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}\begin{cases}D_{Mi}(\mathbf{x},\mathbf{y}) & \text{Jeżeli atrybut ciągy}\\ D_H(\mathbf{x},\mathbf{y}) & \text{Jeżeli atrybut symboliczny/binarny}\end{cases}$$

Klasyfikator k najbliższych sąsiadów

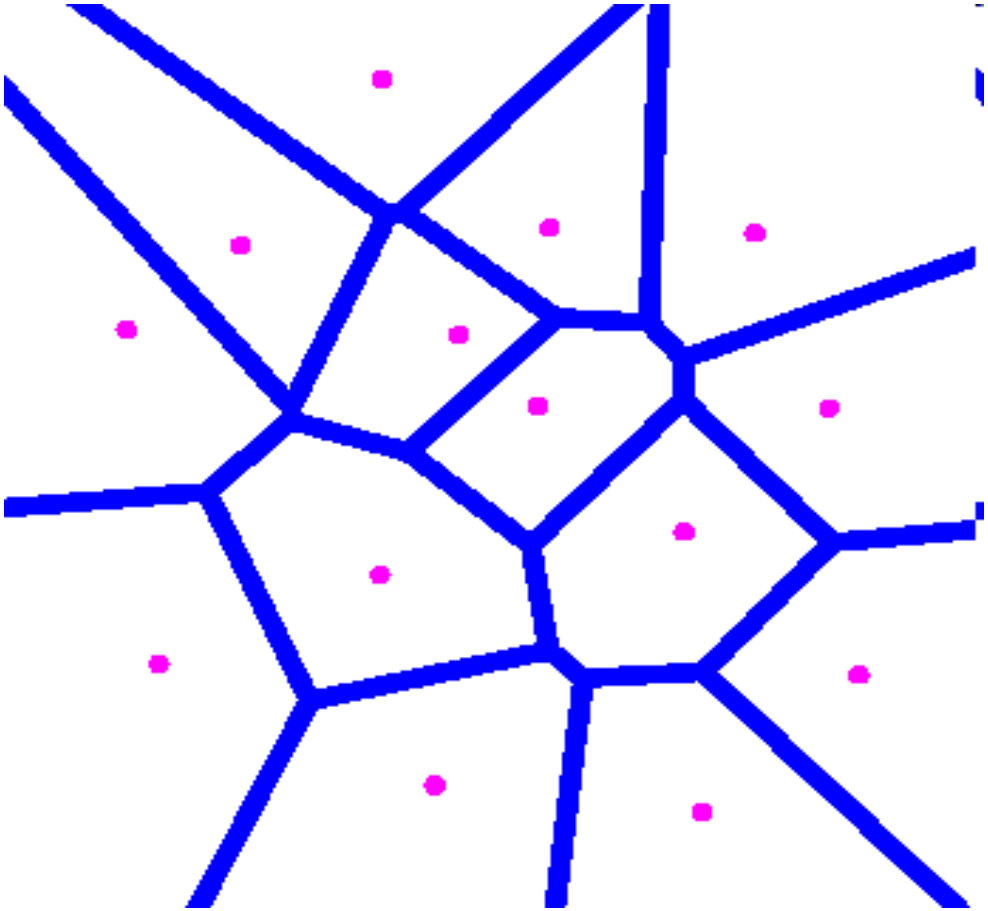
Klasyfikator kNN (k najbliższych sąsiadów)

- Wyznacz odległości wektora testowego **x** do wszystkich przypadków zbioru treningowego.
- Znajdź k najbliższych sąsiadów
- Przeprowadź głosowanie klasy pomiędzy *k* najbliższymi sąsiadami, wybierz klasę najczęściej występującą
- Ewentualne konflikty rozwiąż losowo

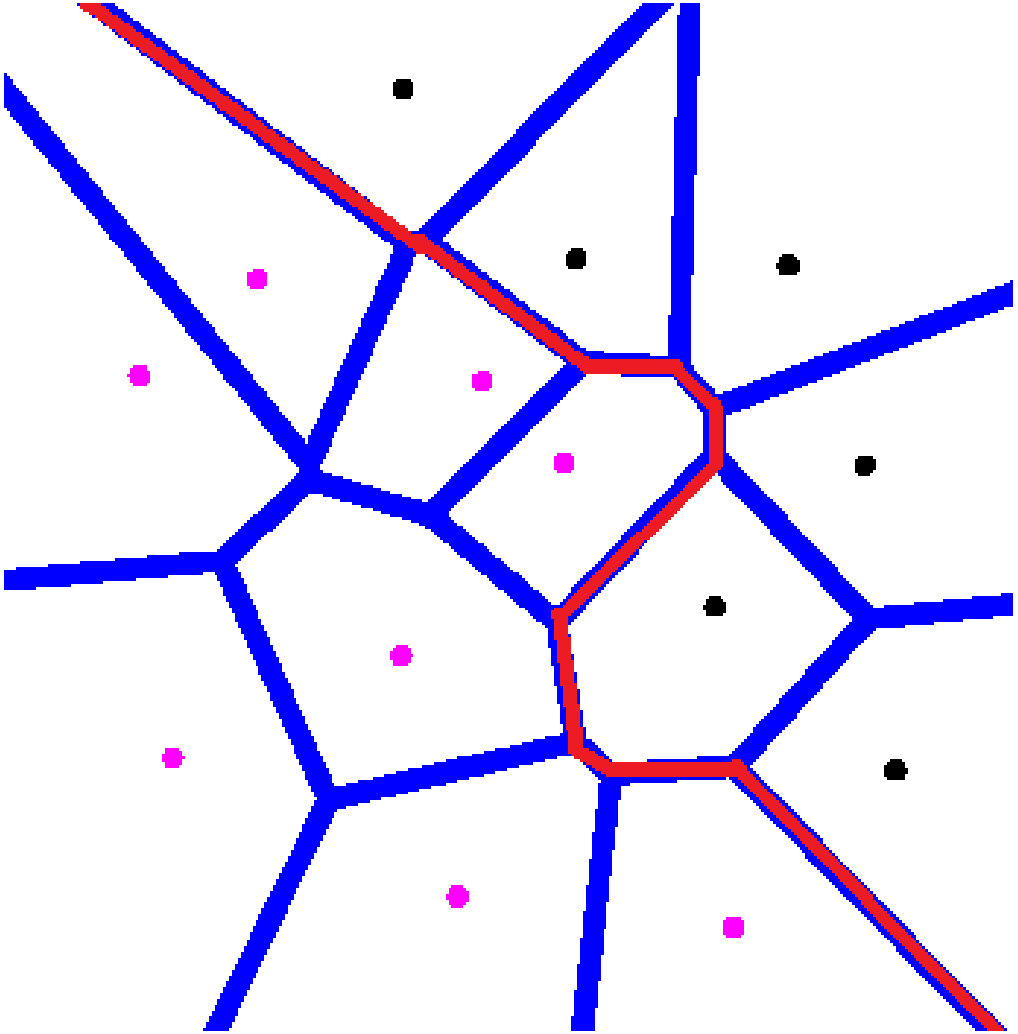
Uwagi na temat kNN

- Dokładność klasyfikatora 1NN na zbiorze treningowym zawsze = 100% !!!
  - Gorzej działa w rzeczywistości, choć i tak dobrze
- W problemach klasyfikacyjnych nigdy nie używaj 2NN, bo w pobliżu granicy decyzji zawsze będzie konflikt podczas głosowania (jeden za, jeden przeciw)
- kNN duży nakład obliczeniowy w przypadku dużych zbiorów treningowych (duża złożoność przy testowaniu)

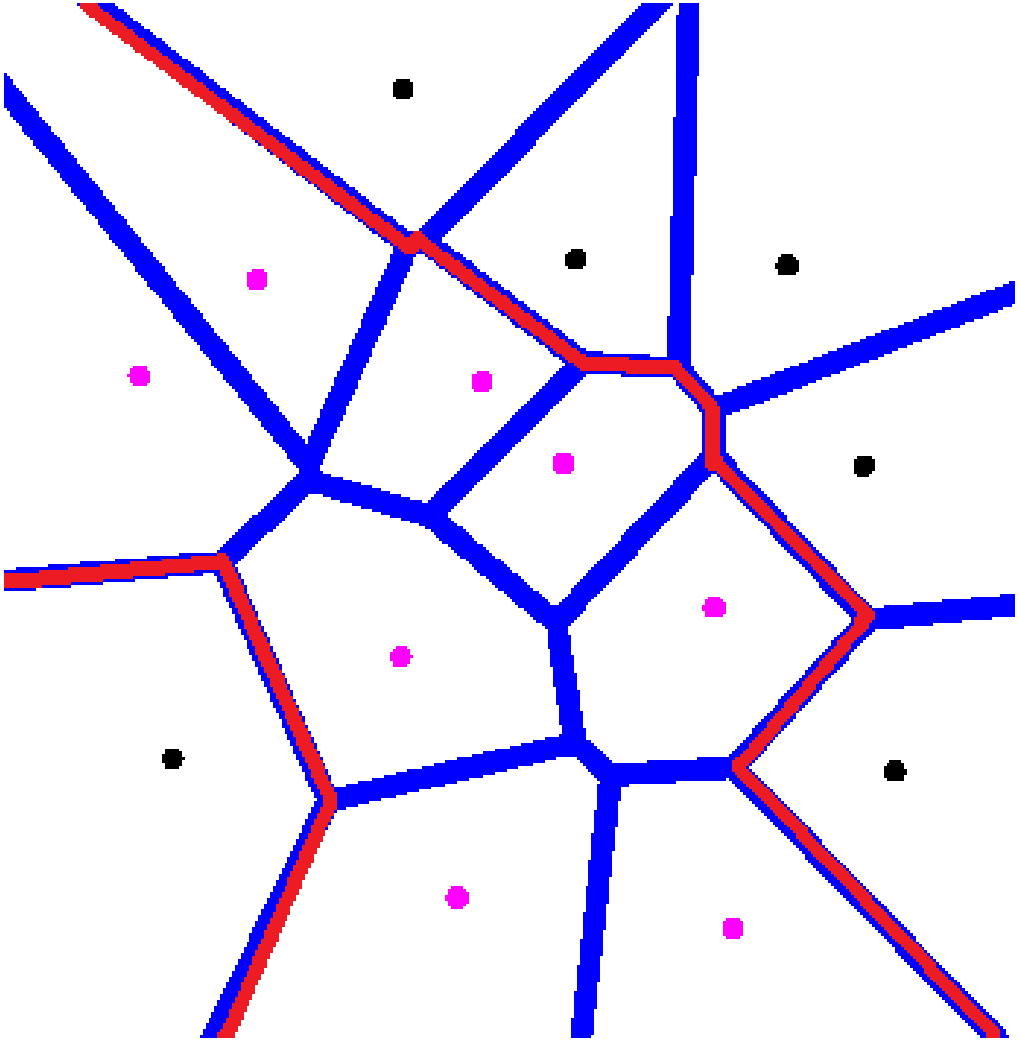
Obszary Voronoi



Obszary Voronoi / Przykład 1NN

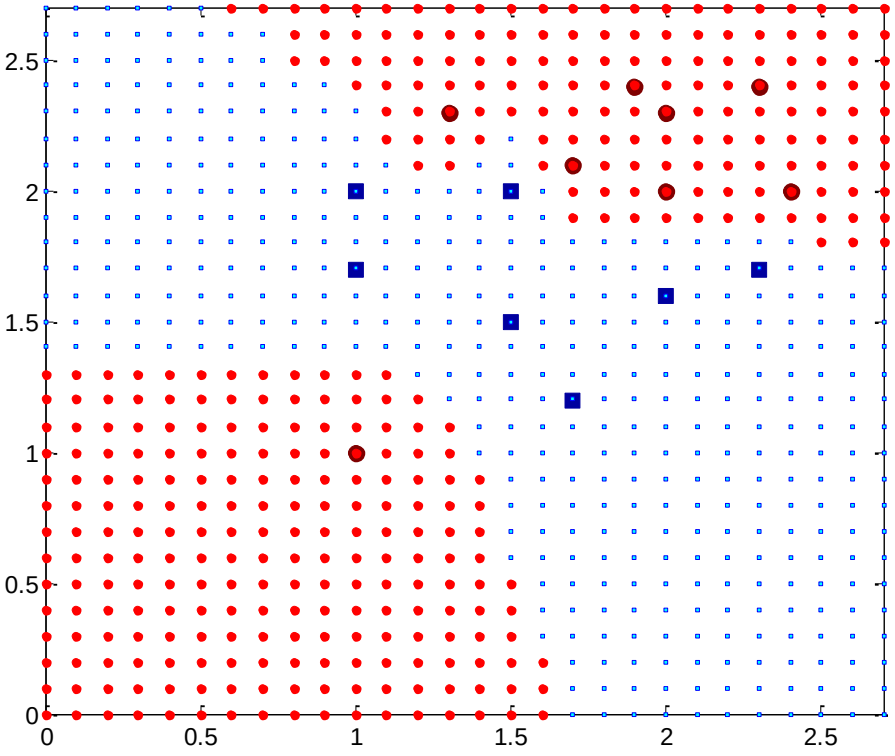


Wada 1NN

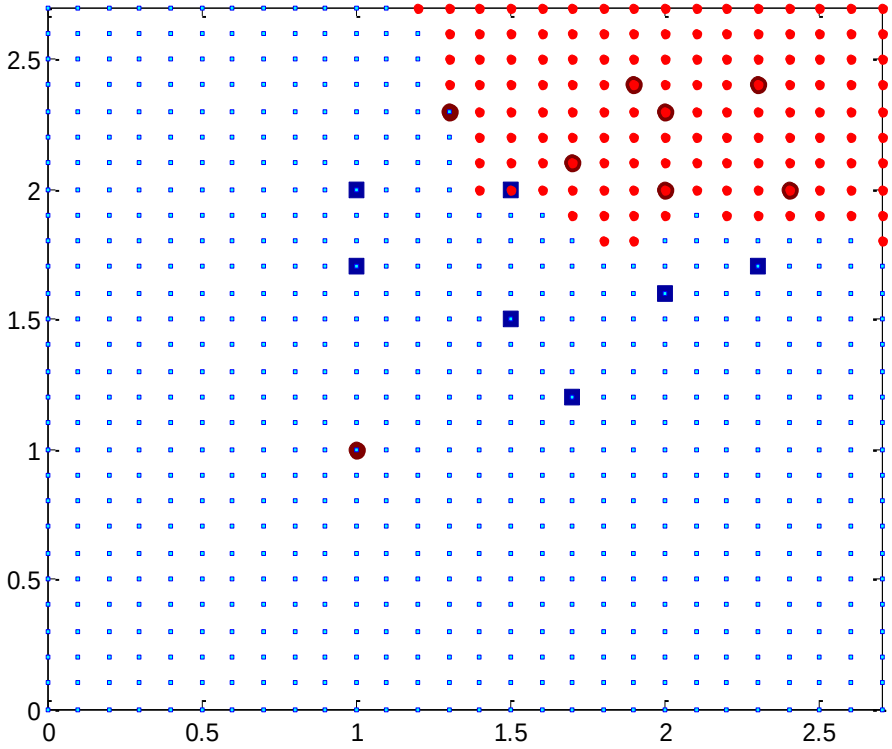


Przykład kNN

1NN



3NN

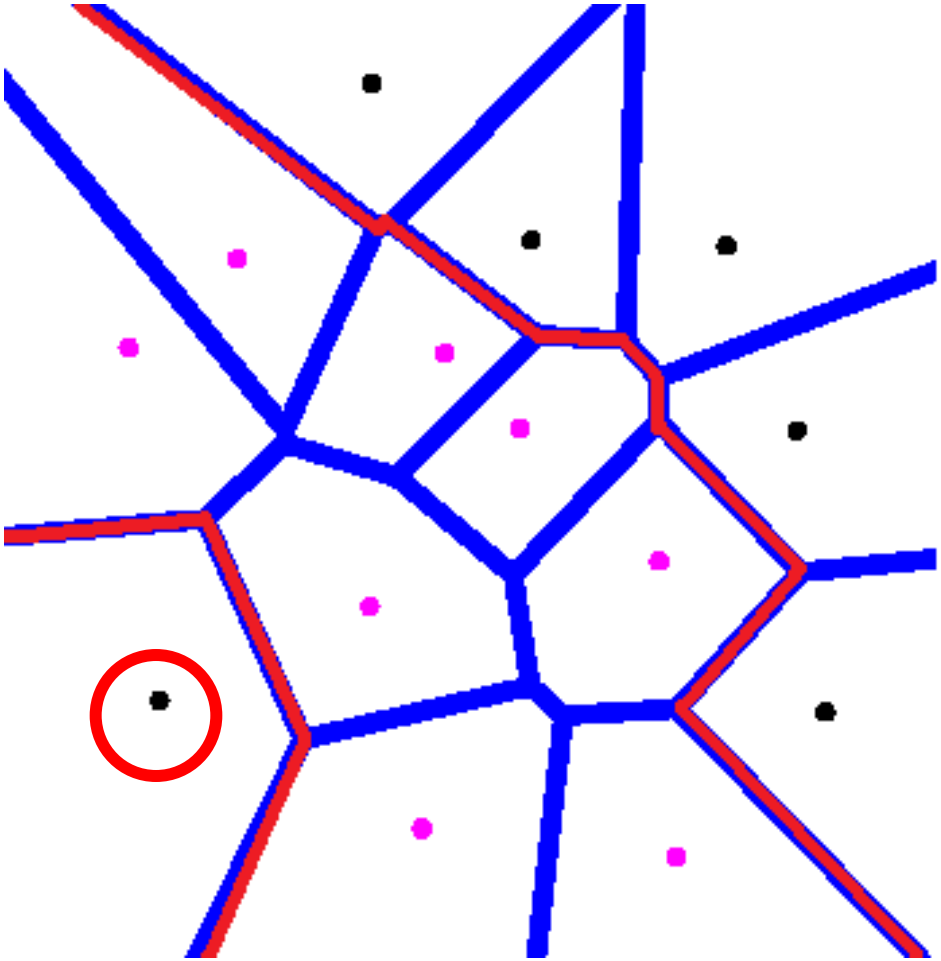


Rozszerzenie kNN – wybór wektorów referencyjnych

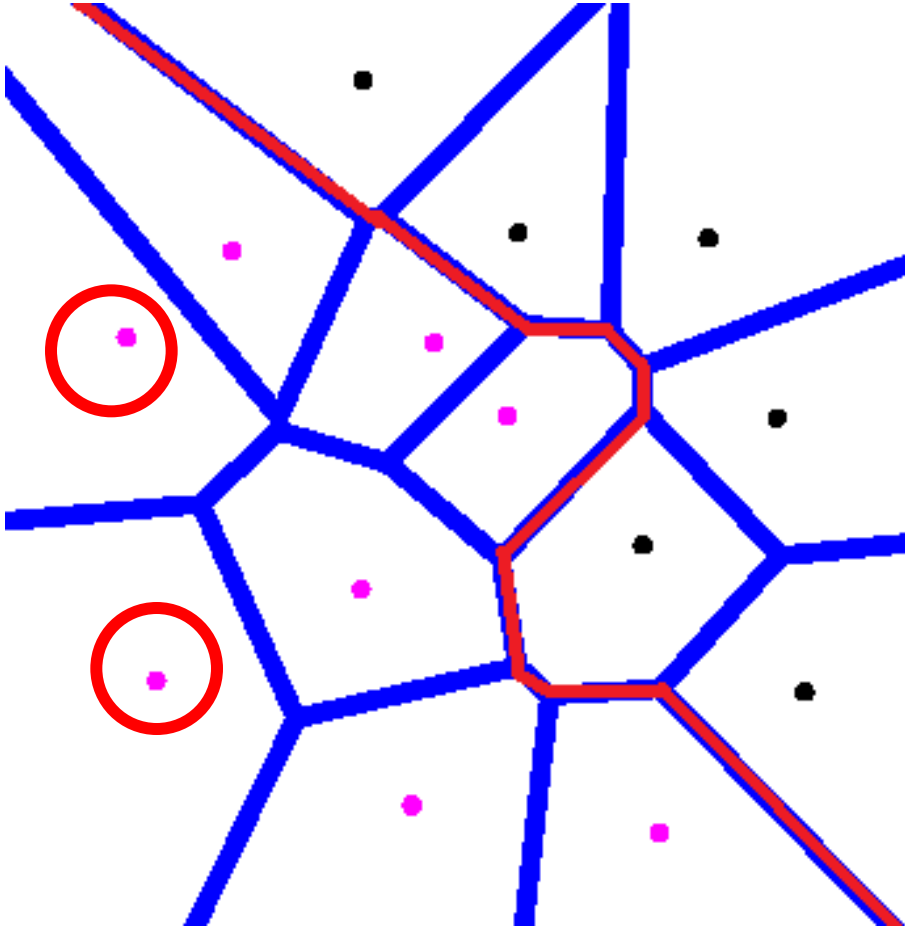
Algorytm kNN można usprawnić poprzez:

- usunięcie ze zbioru prototypów (przechowywanych wektorów) przypadki błędne lub nietypowe
- Usunąć wektory leżące daleko od granicy gdyż one nie biorą czynnego udziału w procesie podejmowania decyzji

Przykład



Przypadki błędne



Wektory nieistotne

Algorytm ENN

- Autor – Wilson. Metoda ta usuwa wszystkie wektory stanowiące szum w zbiorze danych treningowych. Dla każdego wektora wyznaczanych jest *k* najbliższych sąsiadów spośród zbioru uczącego, które użyte są do głosowania. Jeżeli wynikiem głosowania *k* sąsiadów jest błędna klasa, wówczas wektor taki zostaje oznaczony do usunięcia P = T\$\underline{T}$, gdzie $\underline{T}$ jest zbiorem wektorów przeznaczonych do usunięcia. Rezultatem działania tego algorytmu jest usunięcie wektorów odstających oraz wektorów brzegowych, a jedyną wartością nastawną jest *k* (autor zaleca *k* = 3).

Algorytm ENN

Schemat 1 Schemat algorytmu ENN

Require: $\mathbf{T}$

$m \leftarrow sizeof(\mathbf{T})$;
$rem_i \leftarrow 0$;
for $i = 1 \ldots m$ **do**
  $\bar{C}(\mathbf{x}_i) = k\text{-NN}\ ((\mathbf{T} \setminus \mathbf{x}_i), \mathbf{x}_i)$;
  if $C(\mathbf{x}_i) \neq \bar{C}(\mathbf{x}_i)$ **then**
    $rem_i = 1$;
  end if
end for
for $i = 1 \ldots m$ **do**
  if $rem_i == 1$ **then**
    $\mathbf{T} = \mathbf{T} \setminus \mathbf{x}_i$
  end if
end for
return $\mathbf{P}$

Algorytm RENN i All kNN

- RENN

  Modyfikacją algorytmu ENN jest metoda RENN (ang. repeated ENN), gdzie algorytm ENN jest wielokrotnie powtarzany aż do momentu, w którym żaden z wektorów nie jest już usuwany w wyniku działania algorytmu ENN.

- All kNN

  algorytm *All k*-NN polega na porównywaniu wyników dla różnych wartości parametru *k*.

Algorytm RENN

Schemat 2 Schemat algorytmu RENN

Require: $\mathbf{T}$ $flaga \leftarrow$ **true**
  while falga **do**
    $flaga \leftarrow$ **false**
    $\bar{\mathbf{P}} = \mathrm{ENN}(\mathbf{P}, \mathbf{T})$
    if $\bar{\mathbf{P}} \neq \mathbf{P}$ **then**
      $flaga \leftarrow$ **true**
    end if
  end while
  return $\mathbf{P}$

Algorytm CNN

- Metoda ta należy do grupy przyrostowych. Rozpoczyna ona od losowo wybranego wektora jako prototypu P, następnie w pętli klasyfikuje pozostałe przypadki i jeżeli któryś zostaje błędnie sklasyfikowany przez aktualny zbiór prototypów jest on do niego dodawany $P = P \cup x$. Procedura ta jest powtarzana aż wszystkie wektory zostają sklasyfikowane poprawnie.

Algorytm CNN

Schemat 3 Schemat algorytmu CNN

Require: $\mathbf{T}$
  $m \leftarrow sizeof(\mathbf{T})$
  $\mathbf{p}_1 \leftarrow \mathbf{x}_1$
  $flaga \leftarrow$ **true**
  while flaga **do**
    $flaga \leftarrow$ **false**
    for $i = 1 \ldots m$ **do**
      $\bar{C}(\mathbf{x}_i) = k\text{-NN}\ (\mathbf{P}, \mathbf{x}_i)$
      if $\bar{C}(\mathbf{x}_i) \neq C(\mathbf{x}_i)$ **then**
        $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{x}_i$;
        $\mathbf{T} \leftarrow \mathbf{T} \setminus \mathbf{x}_i$
        $flaga \leftarrow$ **true**
      end if
    end for
  end while
  return $\mathbf{P}$

Inne metody

- RNN (Wilson/Martinez)
- DROP1-5 (Wilson/Martinez)

} Bazują na CNN

- Metody GE oraz RNG – bazują bezpośrednio n stworzeniu i oczyszczeniu diagramu Voronoi
- Metody selekcji losowej/genetycznej/wspinaczki itp.
- Metody grupowania danych (klasteryzacji)
- Algorytm częściowo nadzorowanej klasteryzacji CFCM

Sieci LVQ a 1NN

- Usprawnienie 1NN – redukcja liczby wzorców
  - Zamiast wykorzystywać cały zbiór danych treningowych w 1NN, można wziąć tylko wybrane – najważniejsze przypadki, ale które?
  - Rozwiązanie:

sieci typu LVQ

Sieci LVQ szukają metodami optymalizacyjnymi najlepszego położenia wektorów wzorcowych

LVQ pojęcia

- Wektory kodujące (wzorce do których będziemy porównywali nasz wektor testowy)
- Miara odległości – określa stopień podobieństwa do wektora testowego, w praktyce najczęściej odległość Euklidesa

Uczenie sieci LVQ1

Algorytm uczenia

- Określ liczbę wektorów kodujących
- Wylosuj położenie każdego z wektorów kodujących p_i
- Dla danego wzorca uczącego x_j znajdź najbliższy wzorzec kodujący,
- Dokonaj aktualizacji położenia wzorca zgodnie z zależnością

$$\mathbf{p}_i(z+1) = \mathbf{p}_i(z) - \alpha(z)(\mathbf{x}_j - \mathbf{p}_i(z))$$
$$\mathbf{p}_i(z+1) = \mathbf{p}_i(z) + \alpha(z)(\mathbf{x}_j - \mathbf{p}_i(z))$$

- Symbol (+) występuje gdy obydwa wzorce są z tej samej klasy
  – wektor treningowy przyciąga wzorzec kodujący
- Symbol (–) występuje gdy obydwa wzorce są z różnych klas
  – wektor treningowy odpycha wzorzec kodujący
- Dokonaj aktualizacji wsp. uczenia α (z to numer iteracji)

Uczenie sieci LVQ2

Algorytm uczenia

- Dla danego wektora treningowego x_j znajdź dwa najbliższe wektory kodujące
- Jeśli etykiety wektorów kodujących są zgodne z etykietą wektora treningowego aktualizuj położenie najbliższego wektora kodującego (jak LVQ1)
- Jeśli etykiety sąsiednich wektorów kodujących są różne, aktualizuj zgodnie z zależnością LVQ1 obydwa wektory kodujące (przyciągaj wektor zgodny, odpychaj niezgodny)

(Lepsza dokładność niż LVQ1)

LVQ2.1

- Algorytm identyczny z LVQ2 z tą różnicą iż najpierw realizowana jest weryfikacja czy wektor **x** wpada do okna zdefiniowanego jako

$$\min\left(\frac{d_1}{d_2}, \frac{d_2}{d_1}\right) > s \; ; \; s = \frac{1-w}{1+w}$$

- Jeśli tak – rób LVQ2, jeśli nie LVQ1
- Gdzie
  - w – rozmiar okna
  - d1 – odległość wektora **x** od wzorca $\mathbf{p_1}$
  - d2 – odległość wektora **x** od wzorca $\mathbf{p_2}$
  - s – względna wielkość okna

LVQ3

- Identyczny z LVQ2 z tą różnicą iż, jeśli obydwa najbliższe wektory kodujące są z tej samej klasy dokonaj aktualizacji w oparciu o zależność:

$$\mathbf{p}_i(z+1) = \mathbf{p}_i(z) + \varepsilon\alpha(z)(\mathbf{x}_j - \mathbf{p}_i(z))$$

- Gdzie
  - ε - stała zależna od rozmiaru okna **w**

Celem LVQ3 jest nie tylko poprawna klasyfikacja ale również odtworzenie rozkładu danych uczących poprzez równomierny rozkład wektorów wzorcowych

OLVQ – Optimized LVQ

- Uniezależnienie wartości wsp. Uczenia dla każdego wektora kodującego $\alpha_i(z)$ wg. zależności

$$\mathbf{p}_i(z+1) = \mathbf{p}_i(z) - \alpha_i(z)(\mathbf{x}_j - \mathbf{p}_i(z))$$
$$\mathbf{p}_i(z+1) = \mathbf{p}_i(z) + \alpha_i(z)(\mathbf{x}_j - \mathbf{p}_i(z))$$

- aktualizacja wsp. uczenia następuje jedynie dla wektorów kodujących, których położenie zostało zaktualizowane

Cel OLVQ - zapewnienie by wektory rzadko aktualizowane miały szansę wziąć większy udział w procesie uczenia – rzadko aktualizowane mają większy wsp. Uczenia więc mogą bardziej radykalnie zmieniać swoje położenie

Problemy z sieciami LVQ

- Dobór liczby wektorów kodujących
  - Algorytm Dyn LVQ
  - Algorytmy przeszukiwania
  - Algorytm wyścigu
- Inicjalizacja położenia wektorów kodujących
  - Losowa
  - Wstępna klasteryzacja
  - Klasteryzacja nazworowana – algorytm CFCM