Algorytm k-NN

Filip Galas
Dominik Góralczyk

Agenda

1. Wstęp teoretyczny
    a. intuicja, algorytm k-Nearest Neighbors
    b. krótki przegląd kilku algorytmów Nearest Neighbor Search
    c. kernel k-NN
    d. metryki
2. Przykłady w Pythonie (https://github.com/zalon525/mro-knn)

Algorytm

Mamy zbiór treningowy złożony z obiektów (wektorów), dla których znamy ich klasy (w przypadku klasyfikacji) lub wartości (w przypadku regresji).

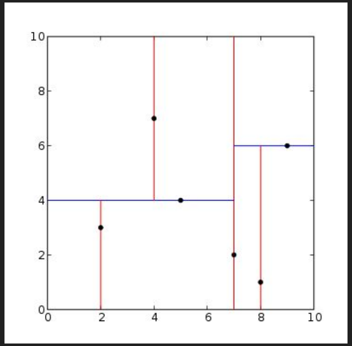
1. Dla danego punktu testowego znajdujemy k najbliższych sąsiadów w zbiorze treningowym.
2. W zależności od rodzaju problemu:
   a. W przypadku klasyfikacji przydzielamy punkt testowy do klasy, do której należy większość k najbliższych sąsiadów (tzw. głosowanie)
   b. W przypadku regresji wartość punktu testowego jest średnią wartości k najbliższych sąsiadów.

k - jest parametrem algorytmu (jest to najczęściej niewielka liczba naturalna)

Jak znajdować najbliższych sąsiadów?

Jest kilka możliwości:

1. Metody dokładne:
    a. Podejście naiwne - dla danego punktu przeszukujemy cały zbiór, aby znaleźć najbliższego sąsiada - złożoność O(n)
    b. Podział przestrzeni - metody polegające na podziale przestrzeni, tak aby ograniczyć przeszukiwany obszar. Np. istnieje metoda wykorzystująca drzewo k-d (zbudowanie drzewa k-d - O(n log n), znalezienie najbliższego sąsiada - średnio O(log n))



Jak znajdować najbliższych sąsiadów

2. Metody aproksymacyjne (parę przykładów):

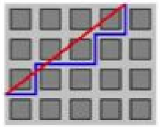
- Locality-sensitive hashing (LSH) - metoda polegająca na przydzielaniu punktów do “kubełków” wg funkcji hashującej skonstruowanej tak, aby dla punktów sobie bliskich zwracała tę samą wartość z dużym prawdopodobieństwem
- Cover tree

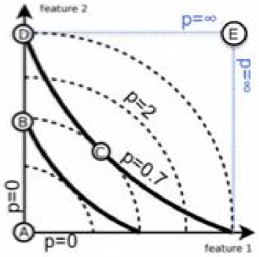
kernel k-NN

$$\max_r \left(\sum_{i=1}^{k} K(D(x, x_{(i)})) I(y_{(i)} = r) \right) \quad .$$

- rectangular kernel $\frac{1}{2} \cdot \mathrm{I}(|d| \leq 1)$
- triangular kernel $(1 - |d|) \cdot \mathrm{I}(|d| \leq 1)$
- Epanechnikov kernel $\frac{3}{4}(1 - d^2) \cdot \mathrm{I}(|d| \leq 1)$
- quartic or biweight kernel $\frac{15}{16}(1 - d^2)^2 \cdot \mathrm{I}(|d| \leq 1)$

- Minkowski distance (p-norm): $D(x,x') = \sqrt[p]{\sum_d |x_d - x'_d|^p}$
  - p=2: Euclidian
  - p=1: Manhattan
  - p=0: Hamming ... logical AND
  - $p=\infty$: $\mathbf{max}_d\ |x_d - x'_d|$ ... logical OR





Metrics intended for real-valued vector spaces:

identifier	class name	args	distance function
"euclidean"	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
"manhattan"	ManhattanDistance	•	`sum(|x - y|)`
"chebyshev"	ChebyshevDistance	•	`max(|x - y|)`
"minkowski"	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
"wminkowski"	WMinkowskiDistance	p, w	`sum(w * |x - y|^p)^(1/p)`
"seuclidean"	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
"mahalanobis"	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

Metrics intended for two-dimensional vector spaces: Note that the haversine distance metric requires data in the form of [latitude, longitude] and both inputs and outputs are in units of radians.

identifier	class name	distance function
"haversine"	HaversineDistance	**2 arcsin(sqrt(sin^2(0.5*dx)** • cos(x1)cos(x2)sin^2(0.5*dy)))

Metrics intended for integer-valued vector spaces: Though intended for integer-valued vectors, these are also valid metrics in the case of real-valued vectors.

identifier	class name	distance function
"hamming"	HammingDistance	`N_unequal(x, y) / N_tot`
"canberra"	CanberraDistance	`sum(|x - y| / (|x| + |y|))`
"braycurtis"	BrayCurtisDistance	`sum(|x - y|) / (sum(|x|) + sum(|y|))`