A Neighborhood-Based Clustering by Means of the Triangle Inequality

Dokumentacja końcowa

Artur G.

09.06.2021

Spis treści

1. Wprowadzanie i definicja problemu

Grupowanie danych jest jednym z ważniejszych zadań zarówno sztucznej inteligencji jak i eksploracji danych (ang. data mining). Jedną z istotniejszych grup algorytmów grupowania danych są algorytmy oparte na grupowaniu gęstościowym. Z omawianych na wykładzie można wyróżnić takie algorytmy jak DBSCAN (oparty na stałym otoczeniu espilonowym) lub NBC (oparty na *k* najbliższych sąsiadach). Jednakże, algorytmy te, w swych podstawowych formach (np. NBC oparte na R-drzewie), są niewydajne w przypadku danych wielowymiarowych.

W takich przypadkach z pomocą przychodzi własność nierówności trójkąta i metody z niej wynikające. Jedną z takich metod jest algorytm *TI-k-Neighborhood-Index*[1], który wspiera standardową metodę NBC w celu zwiększenia wydajności na danych wielowymiarowych.

2. Charakterystyka algorytmu

Implementowane rozwiązanie będzie odnosiło się do algorytmu NBC wspieranego o *TI-k-Neighborhood-Index*. Dlatego też, rozwiązanie to można podzielić na dwa etapy:

1. wyznaczenie k-sąsiedztwa (zbudowanie indeksu) dla każdego punktu przy pomocy algorytmu *TI-k-Neighborhood-Index*,
2. implementacja NBC, które będzie wykorzystywać indeks wyznaczony w poprzednim kroku.

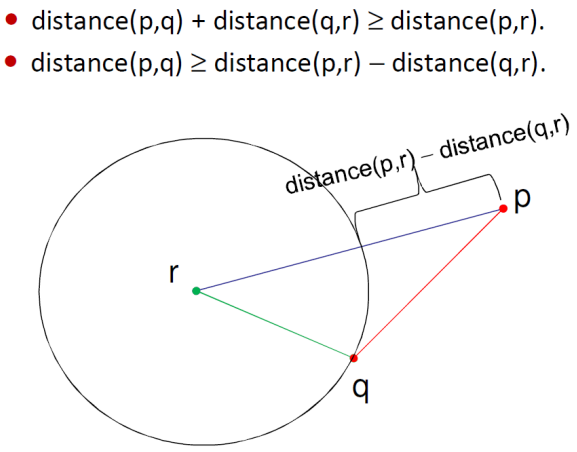
2.1. *TI-k-Neighborhood-Index*

Algorytm ten wykorzystuję nierówność trójkąta w celu ograniczeń nakładu obliczeń wynikającego z potrzeby wynaczynienia odległości wszystkich punktów względem wszystkich punktów. Przybliżony przebieg algorytmu prezentuję się następująco:

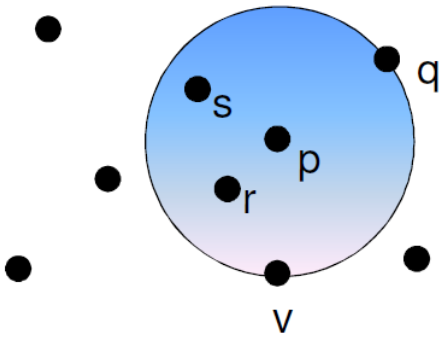
1. wyznacz dystans wszystkich punktów do punktu referencyjnego o wszystkich współrzędnych 0,
2. posortuj punkty względem wyznaczonych odległości,
3. dla każdego punktu *p*:
   a) wyznacz *k* punktów będących początkowymi kandydatami na najbliższych sąsiadów punktu *p*,
   b) wyznacz *Eps* na podstawie początkowych kandydatów (max z odległości kandydatów do punktu *p*). Wyznaczone *Eps* jest promieniem wewnątrz, którego na pewno znajduje się *k* najbliższych sąsiadów punktu *p*, wynika to nierówności trójkąta,
   c) sprawdzaj pozostałe punkty (poprzedzające i następujące względem wyznaczonych kandydatów) dopóki różnica odległości sprawdzanego punktu i punktu *p* do punktu referencyjnego nie będzie większa od *Eps* (lub *-Eps*, zależnie od tego czy punkt jest następujący czy poprzedzający),

d) zwróć k najbliższych sąsiadów.

Jest to pobieżny opis tego algorytmu, bardziej szczegółowe wyjaśnienie wraz z pseudokodem znajduję się w artykule [1].



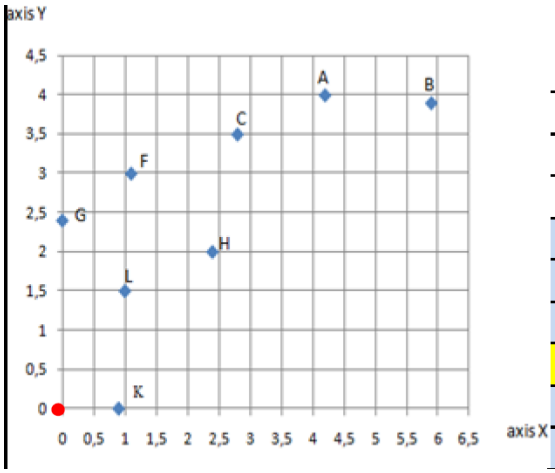
Rysunek 2.1. Nierówność trójkąta dla dowolnych punktów. (źródło: wykład)



Rysunek 2.2. Przykład działania *TI-k-Neighborhood-Index*. (źródło: wykład)

2.2. NBC

Algorytm NBC jest algorytmem grupowania opartym na gęstości. NBC korzysta z współczynnika NDF do wyznaczania gęstości podprzestrzeni. Wyznacznik ten jest rozumiany jako stosunek liczności odwrotnego k-sąsiedztwa do liczności k-sąsiedztwa. Odwrotne k-sąsiedztwo jest definiowane jako zbiór punktów w zbiorze wejściowym, dla których punkt, dla którego wyznaczany jest współczynnik, jest k-sąsiadem. Jeśli wartość tego współczynnika jest większa lub równa 1 to punk pełni rolę punktu rdzeniowego. Punkt rdzeniowy jest interpretowany jako ziarno, które wraz ze swoim k-sąsiedztwem reprezentuje gęstą przestrzeń, którą można uznać za grupę lub część grupy. Kiedykolwiek punkt rdzeniowy jest dołączany do grupy, wszystkie punkty w jego k-sąsiedztwie także są włączane do tej grupy, chyba, że wcześniej zostały przypisane do innej grupy.



Uporządkowany zbiór punktów D;
Eps' = 1.55

q	X	Y	distance(q,r)	distance(C,q)
K	0,9	0,0	0,9	
L	1,0	1,5	1,8	
G	0,0	2,4	2,4	
H	2,4	2,0	3,1	1.55
F	1,1	3,0	3,2	1.77
C	2,8	3,5	4,5	
A	4,2	4,0	5,8	1.49
B	5,9	3,9	7,1	

Rysunek 2.3. Grupowanie oparte na *k*-sąsiedztwie. (źródło: wykład)

3. Opis implementacji

Wybrane zadanie zostało zaimplementowane w języku Python. Rozwiązanie to można podzielić na dwie główne sekcji. mianowicie:

- Wyznaczanie k-sąsiedztwa dla każdego punktu tj. budowanie indeksu,
- Grupowanie punktów na podstawie sąsiedztwa.

Kod źródłowy projektu dostępny jest w serwisie Github na publicznym repozytorium *agrud-kow / nbc-knn-ti*[2]

3.1. Wyznaczanie k-sąsiedztwa

Jest to główny temat tergo projektu, ponieważ to ten element całego procesu grupowania widnieje tu jako ulepszenie. Warto nadmienić, iż zaimplementowano dwie wersie wyznaczanie k-sąsiedztwa:

- opartą o wyznaczanie k-sąsiedztwa przy wykorzystaniu nierówności trójkąta,
- standardowa, opartą na wyznaczeniu wszystkich odległości między punktami.

3.1.1. Wyznaczanie k-sąsiedztwa na podstawie nierówności trójkąta

Metoda ta została zaimplementowana bazując na algorytmie przedstawiony w publikacji [1] i opisanym w poprzednich rozdziałach. Wszystkie opisane tam metody zostały zenkapsulowane do postaci metod klasy o nazwie *TIkNeighborhoodIndex*. Skróconą implementacje ów klasy przedstawiono poniżej.

```
class TIkNeighborhoodIndex():

  def __init__(self, data: np.array, dimensions: int, k: int = 3) ->
   ↪ None:
    ...
```

```
def ti_k_neighborhood(self, p_idx: int) -> np.array:
  ...

def preceding_point(self, idx) -> Tuple[int, bool]:
  ...

def following_point(self, idx) -> Tuple[int, bool]:
  ...

def find_first_kcn_fb(
  ...

def find_first_kcn_b(
  ...

def find_first_kcn_f(
  ...

def verify_k_condidate_neighbours_backward(self, knn: KNN, p_idx: int,
 ↪  b_idx: int, backwardSearch: bool,
  ...


def verify_k_condidate_neighbours_forward(self, knn: KNN, p_idx: int,
 ↪  f_idx: int, forwardSearch: bool,
  ...

def get_idx_from_dist(self, idx) -> int:
  ...

def calc_real_distance(self, idx_1, idx_2) -> float:
 ...

def create_est_dist_list(self) -> KNN:
  ...
```

3.1.2. Wyznaczanie k-sąsiedztwa metodą standardową

W tym przypadku kod takiej funkcjonalności jest znacznie mniej skomplikowany, ponieważ sprowadza się on do podwójnie zagnieżdżonej pętli iterującej po zbiorze danych. W pierwszej iteracji wyznaczamy dla danego punktu odległości wszystkich innych punktów

do niego po czym wybieramy tylko k-najbliższych sąsiadów (lub więcej, jeśli jakieś skrajne punkty są równoodległe). Implementacje tego algorytmu przedstawiono poniżej.

```
def k_neighbourhood(data: np.ndarray, k: int) -> Tuple[KNNS, R_KNNS]:
  knns: List[List[Tuple[float, int]]] = [list() for _ in
  ↪  range(len(data))]
  r_knns: List[List[int]] = [list() for _ in range(len(data))]

  for idx1, v1 in enumerate(data):
    neighbour_candidates = []
    for idx2, v2 in enumerate(data):
      if idx1 != idx2:
        dist = distance(v1, v2)
        neighbour_candidates.append((dist, idx2))
    neighbour_candidates.sort(key=lambda t: t[0])
    eps = neighbour_candidates[:k][-1][0]

    neighbours = []
    for nc in neighbour_candidates:
      if nc[0] > eps:
        break
      neighbours.append(nc)

    knns[idx1] = neighbours
    for nc in knns[idx1]:
      r_knns[nc[1]].append(idx1)

  return knns, r_knns
```

3.2. Algorytm NBC

Algorytm NBC został zaimplementowany w postaci pętli, która iterując po danych sprawdza czy nie są one punktami gęstymi (dokładny opis znajduje się w rozdziale 2.2) wykorzystując przy tym k-sąsiedztwa i odwrotne k-sąsiedztwa wyznaczone przy pomocy jednej z dwóch opisanych wyżej metod. Fragment implementacji znajduje się poniżej.

```
def nbc(data: np.array, dimensions: int, k: int, index_type: str = '')
 ↪  -> CLUSTER:
  clusters: CLUSTER = [EMPTY_CLUSTER] * len(data)

  if index_type == 'ti-kn':
    tikni = TIkNeighborhoodIndex(data, dimensions, k)
```

```
    knns, r_knns = tikni.run()
  elif index_type == 'kn':
    knns, r_knns = k_neighbourhood(data, k)
  else:
    raise AttributeError('Index type `{}` does not exist. Use `ti-kn` or
     ↪  `kn`.'.format(index_type))

  ndf = calc_ndf(knns, r_knns)

  current_cluster_id = 0
  for idx, _ in enumerate(data):
    if has_cluster(idx, clusters) or not is_dense_point(idx, ndf):
      continue
    clusters[idx] = current_cluster_id
    dense_points = set()

    for n in knns[idx]:
      n_idx = n[1]
      clusters[n_idx] = current_cluster_id
      if is_dense_point(n_idx, ndf):
        dense_points.add(n_idx)

    while dense_points:
      dp = dense_points.pop()
      for n in knns[dp]:
        n_idx = n[1]
        if has_cluster(n_idx, clusters):
          continue
        clusters[n_idx] = current_cluster_id
        if is_dense_point(n_idx, ndf):
          dense_points.add(n_idx)

    current_cluster_id += 1

  return clusters
```

4. Instrukcja obsługi

Główny plik wykonywalny przeznaczony do ugotowania w postaci metody do wyznaczania grupowania znajduje się w katalogu *src* i nosi nazwę *main.py*. Wywołanie ów skryptu potrzebuje do poprawnego działania następujących flag:

- input - ścieżka do pliku wejściowego w postaci pliku csv(z separatorem w postaci ’,’) bez nagłówków,
- output - ścieżka do pliku, w którym zostaną zapisane grupy, do których przydzielono poszczególne punkty (również jest to plik csv),
- k - liczba nalbliższych sąsiadów,
- index_type - typ indeksu, z którego algorytm będzie korzystał. Dostępne indeksy to:
  - ti-kn - metoda wyznaczania indeksów k-sąsiedztwa oparta o nierówność trójkąta,
  - kn - standardowy metoda wyznaczania indeksów k-sąsiedztwa.

Przykładowe wydołanie programu przedstawia się następująco:

```
python3 src/main.py --input=data/test_data_2.csv
 ↪ --output=results/out1.csv  --k=3 --index_type=kn
```

5. Zbiory danych

Do przeprowadzenia testów wydajnościowych wykorzystano następujące zbiory danych:

- Absenteeism at work Data Set[3]
- Human Activity Recognition Using Smartphones Data Set[4]

Zbiory te różnią się znacznie liczbą wymiarów przez powinny dobrze uwidaczniać różnice między algorytmami.

5.1. Charakterystyka zbioru Absenteeism at work

Liczba punktów: 740
Liczba atrybutów: 21

5.2. Charakterystyka zbioru Human Activity Recognition Using Smartphones

Liczba punktów: 10299
Liczba atrybutów: 561

6. Wyniki

W celu weryfikacji wyników zostały przeprowadzone testy wydajnościowe, które polegały na zmierzeniu czasu wykonania metod wyznaczających k-sąsiedztwo. Ograniczone się tylko do tych metod a nie do całego algorytmu, ponieważ implementacja NBC jest niezależna od sposoby wyznaczania k-sąsiedztwa a co za tym idzie nie dostarczyłoby to żadnych wyników poza czasem wykonania zwiększonym o czas wykonania NBC.

6.1. Wyniki dla Absenteeism at work

k	**Czas wykonania(s) - metoda standardowa**	**Czas wykonania(s) - metoda TI-k-Neighborhood**
5	2.665634	8.429350
10	2.655447	11.318147
25	2.678044	14.951043
50	2.732866	16.659003
100	2.79761	18.413207

Tabela 6.1. Wyniki indeksowania k-sąsiedztwa na zbiorze Absenteeism at work

6.2. Wyniki dla Human Activity Recognition Using Smartphones

k	**Czas wykonania(s) - metoda standardowa**	**Czas wykonania(s) - metoda TI-k-Neighborhood**
5	353.631777	3376.365453
10	355.683249	4133.273895
25	366.311537	NA[a]
50	362.894788	NA
100	364.800372	NA

Tabela 6.2. Wyniki indeksowania k-sąsiedztwa na zbiorze Human Activity Recognition Using Smartphones

[a] Brak otrzymanych wyników ze względu na wysoki czas oczekiwania.

6.3. Obserwacje

Jak widać w tabelach 6.1 oraz 6.2 wyniki są absolutnie niespodziewane. Metoda, która z pozoru miała osiągać znacznie lepsze wyniki jest znacząco gorsza od metody standardowej

liczącej wszystkie odległości. W przypadku większego zbioru danych można zaobserwować co najmniej 10 krotny wzrost czasu potrzebnego na wyznaczenie k-sąsiedztwa.

7. Wnioski

Analizując wyniki można dojść do wniosku, że coś poszło nie tak, w końcu algorytm oparty o nierówność trójkąta powinien sobie radzić o wiele lepiej, zwłaszcza w większych zbiorach danych. Przy czym pisząc rosnących mam na myśli większą liczbę przykładów niż wymiarów, ponieważ nie jestem do końca przekonany co do zysków z ów metody w przypadkach zwiększającej się liczby wymiarów, jest to raczej kwestia implementacji wyznaczania wydajnego wyliczania odległości między punkami niż cecha tego algorytmu. Jednym z podejrzeń (zakładając poprawność implementacji) jest fakt implementacji tego rozwiązania w Python-ie co nie daje najlepszych rezultatów w połączeniu ze znacznie większą liczbą metod, które muszą być wywołane wewnątrz algorytmu w porównaniu do standardowego podejścia.

Bibliografia

[1] M. Kryszkiewicz i P. Lasek, „A Neighborhood-Based Clustering by Means of the Triangle Inequality”, eng, w *Intelligent Data Engineering and Automated Learning – IDEAL 2010*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, s. 284–291, ISBN: 9783642153808.

[2] *A Neighborhood-Based Clustering (kNN) by Means of the Triangle Inequality*, 2021. adr.: `https://github.com/agrudkow/nbc-knn-ti`.

[3] *Absenteeism at work Data Set*, 2018. adr.: `https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work#`.

[4] *Human Activity Recognition Using Smartphones Data Set*, 2012. adr.: `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`.

Spis rysunków