MED

Sprawozdanie z Projektu

Temat projektu: Algorytm GSP uogólniony

Mateusz Krakowski

22 maja 2024

Spis treści

1 Wprowadzenie

1.1 Treść zadania

Algorytm GSP

Celem projektu jest zbadanie własności zaimplementowanego przez siebie algorytmu GSP: czas wykonania, uzyskiwane wyniki dla różnych wartości parametrów algorytmu oraz kilku zbiorów wejściowych.

1.2 Wprowadzenie i definicja problemu

W dzisiejszej erze danych, gromadzenie ogromnych ilości informacji stało się powszechne w różnych dziedzinach. Wraz z tym wzrostem danych pojawiła się potrzeba efektywnej analizy sekwencji zdarzeń, by wyciągać wartościowe wnioski. Narzędzia do odkrywania sekwencyjnych wzorców, jak algorytm GSP (Generalized Sequential Pattern), odgrywają kluczową rolę.

Projekt skupia się na analizie sekwencji zdarzeń w dużych zbiorach danych. Szczególnie ważna jest identyfikacja częstych sekwencji zdarzeń lub transakcji, takich jak historia zakupów klientów w sklepach detalicznych czy sekwencje odwiedzanych stron internetowych przez użytkowników. Algorytm GSP pozwala na odkrycie istotnych wzorców zachowań lub preferencji, co może być użyteczne w podejmowaniu decyzji biznesowych, personalizacji ofert czy optymalizacji interfejsów użytkownika. Poprzez jego zastosowanie, dane wejściowe przetwarzane są w postaci sekwencji zdarzeń, ustalane jest minimalne wsparcie, a następnie identyfikowane są sekwencje spełniające to kryterium. Dzięki temu możliwe jest wyróżnienie istotnych wzorców, umożliwiając bardziej szczegółową analizę i trafniejsze decyzje biznesowe. Wsparcie należy rozumieć jako liczbę występowania zjawiska w bazie danych.

2 Charakterystyka algorytmu GSP

Algorytm GSP (Generalized Sequential Pattern) jest klasycznym algorytmem eksploracji danych sekwencyjnych, który został zaprojektowany do znajdowania częstych wzorców w dużych zbiorach danych sekwencyjnych. Algorytm ten jest szczególnie użyteczny w kontekście analizy sekwencji zdarzeń w bazach danych, takich jak transakcje klientów w systemach e-commerce, dane o pacjentach w medycynie, czy logi zdarzeń w systemach informatycznych.

2.1 Główne założenia algorytmu

Algorytm GSP bazuje na kilku kluczowych założeniach:

- **Minimalne wsparcie (min_support)**: Określa minimalny próg, który wzorce muszą spełniać, aby zostać uznane za częste. Jest to stosunek liczby sekwencji zawierających dany wzorzec do całkowitej liczby sekwencji w bazie danych.
- **Częstotliwość wzorców**: Wzorce sekwencyjne są generowane i oceniane na podstawie ich częstotliwości występowania w danych. Algorytm iteracyjnie generuje wzorce o coraz większej długości, zaczynając od wzorców jednowymiarowych.

- **Przeszukiwanie bazy danych**: Baza danych jest przeszukiwana w celu zliczenia liczby wystąpień każdej sekwencji kandydującej. Na tej podstawie obliczane jest wsparcie wzorców.

2.2 Kroki algorytmu

Algorytm GSP[2] składa się z kilku głównych etapów:

1. **Inicjalizacja**: Na początku algorytm identyfikuje wszystkie unikalne elementy w bazie danych i oblicza ich wsparcie. Elementy, które spełniają kryterium minimalnego wsparcia, są uznane za częste wzorce jednowymiarowe.
2. **Generowanie kandydatów**: Na podstawie częstych wzorców jednowymiarowych algorytm generuje kandydatów na wzorce dwuwymiarowe poprzez łączenie par elementów. Proces ten jest powtarzany iteracyjnie, generując wzorce o coraz większej długości.
3. **Przeszukiwanie bazy danych**: Każda sekwencja kandydująca jest oceniana pod kątem jej występowania w bazie danych. Algorytm zlicza liczbę sekwencji w bazie danych, które zawierają danego kandydata.
4. **Filtracja**: Sekwencje kandydujące, które spełniają kryterium minimalnego wsparcia, są dodawane do listy częstych wzorców. Proces generowania kandydatów i przeszukiwania bazy danych jest kontynuowany do momentu, gdy nie można wygenerować nowych częstych wzorców.

2.3 Zalety i wady algorytmu GSP

Algorytm GSP ma kilka zalet i wad, które warto uwzględnić przy jego zastosowaniu:

- **Zalety**:
  - Skuteczność w znajdowaniu częstych wzorców w dużych zbiorach danych sekwencyjnych.
  - Możliwość ustawienia progu minimalnego wsparcia, co pozwala na kontrolę liczby generowanych wzorców.
- **Wady**:
  - Wysokie wymagania obliczeniowe i pamięciowe, szczególnie dla dużych zbiorów danych i długich sekwencji.
  - Potrzeba wielokrotnego przeszukiwania bazy danych, co może być czasochłonne.

3 Opis implementacji

Całość algorytmu została zrealizowana w postaci klasy `GSPMK` dostępnej w pliku `gspmk.py`.

3.1 Konstruktor klasy

Konstruktor klasy `GSPMK` przyjmuje dwa argumenty: `min_support` oraz opcjonalny argument `debug`. Argument `min_support` określa minimalne wsparcie, jakie muszą spełniać częste wzorce. Jeśli argument `debug` jest ustawiony na `True`, algorytm wypisuje dodatkowe informacje pomocne przy debugowaniu.

```
def __init__(self, min_support, debug=False):
    self.min_support = min_support
    self.database = None
    self.frequent_patterns = []
    self.debug = debug
    if self.debug:
        print("GSPMK Algorithm")
```

3.2 Generowanie kandydatów

Metoda `generate_candidates` generuje sekwencje kandydujące o długości `length` poprzez łączenie wzorców o długości `length-1`.

```
def generate_candidates(self, patterns, length):
    """
    Generate candidate sequences of length 'length' by joining patterns of length 'lengt
    Ensure no duplicate elements within a candidate sequence.
    """
    candidates = []
    for i in range(len(patterns)):
        for j in range(len(patterns)):
            if i != j and patterns[i][:length-2] == patterns[j][:length-2]:
                candidate = patterns[i] + [patterns[j][-1]]
                if len(set(candidate)) == length:
                    candidates.append(candidate)
    return candidates
```

3.3 Przeszukiwanie bazy danych

Metoda `scan_database` skanuje bazę danych w celu zliczenia wystąpień każdej sekwencji kandydującej.

```
def scan_database(self, candidates):
    support_count = {}
    for seq in self.database:
        for candidate in candidates:
            if all(item in seq for item in candidate):
                support_count[tuple(candidate)] =
                support_count.get(tuple(candidate), 0) + 1
```

```
    frequent_patterns = []
    for seq, count in support_count.items():
        support = count / len(self.database)
        if support >= self.min_support:
            frequent_patterns.append((list(seq), support))

    return frequent_patterns
```

3.4 Dopasowanie modelu

Metoda `fit` dopasowuje model GSP do bazy danych, aby znaleźć częste sekwencje. Proces rozpoczyna się od wygenerowania częstych sekwencji 1-elementowych, a następnie iteracyjnie generowane są sekwencje o większej długości aż do osiągnięcia zadanego `depth`.

```
def fit(self, database, depth=5):
    self.database = database
    self.frequent_patterns = []

    unique_items = set(item for seq in self.database for item in seq)
    for item in unique_items:
        support_count = sum(1 for seq in self.database if item in seq)
        support = support_count / len(self.database)
        if support >= self.min_support:
            self.frequent_patterns.append(([item], support))

    for length in range(2, depth + 1):
        candidates = self.generate_candidates([pattern[0] for
        pattern in self.frequent_patterns], length)
        frequent_patterns = self.scan_database(candidates)
        if not frequent_patterns:
            break
        self.frequent_patterns.extend(frequent_patterns)
```

3.5 Pobieranie częstych wzorców

Metoda `get_frequent_patterns` zwraca znalezione przez algorytm częste sekwencje.

```
def get_frequent_patterns(self):
    return self.frequent_patterns
```

3.6 Eksportowanie częstych wzorców

Metoda `export_frequent_patterns` eksportuje częste sekwencje do pliku CSV o podanej nazwie `filename`.

```
def export_frequent_patterns(self, filename):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        for pattern, support in self.frequent_patterns:
            writer.writerow([pattern, support])
```

4 Instrukcja użytkownika

Klasa `GSPMK` została zaprojektowana, aby umożliwić łatwe wykrywanie częstych sekwencji w zbiorach danych sekwencyjnych. Poniżej znajduje się instrukcja krok po kroku, jak zaimportować tę klasę i jak z niej korzystać.

4.1 Importowanie klasy

Aby skorzystać z klasy `GSPMK`, należy zaimportować ją z pliku `gspmk.py`. Zakładając, że plik ten znajduje się w tym samym katalogu co nasz skrypt, możemy zaimportować klasę w następujący sposób:

```
from gspmk import GSPMK
```

4.2 Tworzenie instancji klasy

Po zaimportowaniu klasy możemy utworzyć jej instancję. Konstruktor klasy przyjmuje dwa argumenty:

- `min_support`: Minimalne wsparcie wymagane, aby wzorzec został uznany za częsty.
- `debug` (opcjonalny): Flaga, która włącza tryb debugowania (domyślnie `False`).

Przykład:

```
gsp = GSPMK(min_support=0.1, debug=True)
```

4.3 Dopasowanie modelu do bazy danych

Aby znaleźć częste sekwencje w bazie danych, należy użyć metody `fit`. Metoda ta przyjmuje dwa argumenty:

- `database`: Lista sekwencji, które mają być analizowane.
- `depth` (opcjonalny): Maksymalna długość sekwencji do rozważenia (domyślnie 5).

Przykład:

```
database = [['A', 'B', 'C'], ['A', 'C'], ['B', 'C'], ['A', 'B', 'C', 'D']]
gsp.fit(database, depth=3)
```

4.4 Pobieranie częstych wzorców

Po dopasowaniu modelu do bazy danych możemy pobrać znalezione częste sekwencje za pomocą metody `get_frequent_patterns`:

```
frequent_patterns = gsp.get_frequent_patterns()
print(frequent_patterns)
```

4.5 Eksportowanie częstych wzorców do pliku CSV

Częste wzorce można eksportować do pliku CSV za pomocą metody `export_frequent_patterns`. Metoda ta przyjmuje jeden argument:

- `filename`: Nazwa pliku, do którego mają zostać zapisane wzorce.

Przykład:

```
gsp.export_frequent_patterns('frequent_patterns.csv')
```

4.6 Pełny przykład użycia

Poniżej przedstawiono pełny przykład użycia klasy `GSPMK`, od importu po eksport wyników do pliku CSV:

```
from gspmk import GSPMK

# Utworzenie instancji klasy z minimalnym wsparciem 0.1 i trybem debugowania włączonym
gsp = GSPMK(min_support=0.1, debug=True)

# Przykładowa baza danych sekwencyjnych
database = [['A', 'B', 'C'], ['A', 'C'], ['B', 'C'], ['A', 'B', 'C', 'D']]

# Dopasowanie modelu do bazy danych
gsp.fit(database, depth=3)

# Pobranie częstych wzorców
frequent_patterns = gsp.get_frequent_patterns()
print(frequent_patterns)

# Eksportowanie częstych wzorców do pliku CSV
gsp.export_frequent_patterns('frequent_patterns.csv')
```

Ten pełny przykład ilustruje cały proces korzystania z klasy `GSPMK` do analizy sekwencyjnej, od inicjalizacji, przez dopasowanie modelu, po eksport wyników.

5 Charakterystyka wykorzystywanych zbiorów danych

Do testów wykorzystano zmodyfikowane zbiory danych dostępne w plikach database1_small.csv i database2_small.csv. Są to skrócone zbiory danych dostępne na podanej przez klienta stronie [1]. Skrócono te zbiory danych, z uwagi na to, że moja implementacja algorytmu w pythonie narzuca duże wymagania obliczeniowe.

Zbiór database1_small składa się z 295 rzędów danych. Każdy z tych rzędów zawiera zestaw liczb, które reprezentują konkretne elementy w analizowanym zbiorze danych. Poniżej przedstawiono przykładowy fragment tego zbioru:

```
...
32,57,145,164,225,368,494,516,658,682,744,782,914,937
...
```

Zbiór database2_small składa się z 500 rzędów danych. Różni on się od database1_small tym, że na rząd przypada o wiele więcej elementów, około 2.5 razy więcej na każdy z rzędów. Poniżej przedstawiono przykładowy fragment tego zbioru:

```
...
53,55,98,159,192,322,332,402,412,413,424,430,450,480,526,538,569,
571,572,598,666,672,694,701,797,809,820,826,897,904,
928,943,952,956,992

...
```

6 Wyniki eksperymentów pokazujących właściwości proponowanego rozwiązania

6.1 Wykryte wzorca i czas działania dla pierwszego zbioru danych

Min_support	Liczba częstych wzorców	Czas (sekundy)
0.01	9932	413.12
0.02	246	85.19
0.03	89	16.67
0.04	40	3.37
0.05	17	0.62
0.06	6	0.11
0.08	1	0.05
0.10	0	0.05

Tabela 1: Liczba częstych wzorców i czas wykonania dla różnych wartości min_support dla zbioru database1_small.csv

Wykryte wzorce można podejrzeć w odpowiednich plikach w katalogu:

`database1_small_different_support`

6.2 Wykryte wzorca i czas działania dla pierwszego zbioru danych

Min_support	Liczba częstych wzorców	Czas (sekundy)
0.02	193366	4539.48
0.03	1821	1051.33
0.04	704	605.58
0.05	376	371.71
0.06	263	219.87
0.08	142	76.36
0.10	89	30.71

Tabela 2: Liczba częstych wzorców i czas wykonania dla różnych wartości min_support dla zbioru database2_small.csv

Podobnie jak przy poprzedniej bazie danych, wykryte wzorce można podejrzeć w odpowiednich plikach w katalogu:

`database2_small_different_support`

7 Wnioski

Podsumowując, analiza wykazała, że liczba wykrytych częstych wzorców gwałtownie wzrasta wraz ze zmniejszaniem wartości `min_support`. Jest to szczególnie widoczne w przypadku drugiego zbioru danych, gdzie przy `min_support` równym 0.02 wykryto aż 193366 wzorców, co znacząco wpłynęło na czas wykonania algorytmu, który wyniósł 4539.48 sekund.

Implementacja algorytmu GSP w Pythonie okazała się być bardzo wymagająca obliczeniowo, co skutkuje koniecznością redukcji rozmiarów zbiorów danych lub zastosowania bardziej wydajnych rozwiązań technologicznych.

Analiza pokazała również, że przy wyższych wartościach `min_support` (0.08 i wyżej), liczba wykrytych wzorców jest znacząco mniejsza, a czas wykonania algorytmu jest stosunkowo krótki. Sugeruje to, że dla praktycznych zastosowań, gdzie zasoby obliczeniowe są ograniczone, wyższe wartości `min_support` mogą być bardziej odpowiednie.

Wnioski te mogą być pomocne w przyszłych analizach i przy podejmowaniu decyzji dotyczących doboru parametrów algorytmu GSP oraz zarządzania zasobami obliczeniowymi.

Podsumowując, algorytm GSP jest potężnym narzędziem do analizy sekwencji, które może dostarczyć cennych informacji na temat wzorców zachowań w danych sekwencyjnych. Jednakże jego zastosowanie wymaga odpowiednich zasobów obliczeniowych i starannego doboru parametrów.

8 Literatura

[1] FIMI Repository. Frequent itemset mining dataset repository, 2024. Accessed: 2024-05-22.

[2] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Advances in Database Technology — EDBT '96*, pages 1–17, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.