**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра програмного забезпечення**

**Звіт**  
про дослідницьку практику за темою магістерської кваліфікаційної роботи

«Аналіз продуктивності роботи алгоритму apriori та його модифікацій»

«Performance analysis of the algorithm apriori and its modifications»

студента (ки) ІІ курсу групи \_\_\_\_

Гаврилюк Андрій Миколайович

(прізвище, ім’я, по-батькові) 

**база практики\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва підприємства)

**термін практики** з «  » до «  »   
 

**Керівники практики**

від кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)     (посада, ПІП)

від бази практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)     (посада, ПІП)   
 

**Керівник магістерської** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис та бали)     (наук.ст., вч.звання, ПІП)

**кваліфікаційної роботи**

**Оцінка**\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Дата\_**\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів 2018

ЗМІСТ

[1. ОПИС МІСЦЯ ПРАКТИКИ 4](#_Toc525760888)

[2. ПОСТАНОВКА ЗАВДАННЯ ДЛЯ ВИКОНАННЯ МАГІСТЕРСЬКОГО ДОСЛІДЖЕННЯ 6](#_Toc525760889)

[3. РЕЗУЛЬТАТИ ВИКОНАННЯ ЗАВДАННЯ ДЛЯ МАГІСТЕРСЬКОГО ДОСЛІДЖЕННЯ 7](#_Toc525760890)

[3.1. Опис відомих модифікацій 7](#_Toc525760891)

[3.2. Алгоритм apriori 9](#_Toc525760892)

[3.3. Опис модифікацій алгоритму для аналізу дорожньо-транспорних пригод 11](#_Toc525760893)

[3.4. Прототипування графічного інтерфейсу користувача 13](#_Toc525760894)

[3.5. Проектування архітектури системи для роботи алгоритму apriori 14](#_Toc525760895)

[4. ВИСНОВКИ 16](#_Toc525760896)

[5. СПИСОК ОПРАЦЬОВАНИХ ДЖЕРЕЛ 17](#_Toc525760897)

[Додаток A. Код РЕАЛІЗАЦІЇ АЛГОРИТМУ 18](#_Toc525760898)

# ОПИС МІСЦЯ ПРАКТИКИ

Edvantis, спочатку відомий як BrainPower Software, обслуговував міжнародні ринки з 2000 року. Наприкінці 2005 року компанія була перейменована в Edvantis Software і значно реорганізовано, щоб прискорити зростання. Сьогодні Edvantis є одним з головних, міжнародних постачальників програмного забезпечення та консалтингу.

Місія компанії — досягнути та утримати найвищий професійний рівень ІТ-послуг задля максимального успіху корпоративних клієнтів у всьому світі; через інновації та цінності стартап-культури сприяти суспільному та індивідуальному розвитку.

Компанія займається як розробкою повних проектів для замовників, так і підтримкою чи доробкою вже готових. Працює з різними мовами програмування та технологіями, такими як Java, C++, C#, .Net, JavaScript, Html та інші. Орієнтована на західний ринок(Європа, США). Має великий досвід в розробці програмного забезпечення для мобільних платформ та веб.

Edvantis відкриває своїм працівникам унікальні можливості для професійного розвитку і втілення найсміливіших проектів та ідей:

* Понад-технологічні сертифікації SWEBOK від IEEE CS (підрозділу найбільшої міжнародної організації інженерів Institute of Electrical and Electronics Engineers). Для України це поки ще нова планка якості знань та оперування наявними інструментами. Програми навчання та сертифікації IEEE CS близько корелюють з програмами Carnegie Mellon University та використовуються такими визнаними лідерами, як Siemens Corporate Development Center. Сертифікований інженер отримує відчутні грошові бонуси, безпосередній кар’єрний ріст,

можливість відвідувати тематичні конференцій та брати участь у викладацько-менторській діяльності.

* ІТ-стартап із Edvantis Startup Launchpad. В рамках цієї програми працівники отримують інфраструктурну, управлінську та фінансову базу для розробки та реалізації власного стартап проекту. Ми допоможемо вам втілити

свою мрію життя. (За умови відсутності потенційних конфліктів з правами на інтелектуальну власність замовників Edvantis).

* Edvantis Intrapreneur — Opportunities Next to You. Кожен працівник компанії може проявити бізнес- та технічну креативність в межах поточного робочого проекту з програмою Edvantis Intrapreneur. У співпраці зі замовником та за допомоги наших досвідених консультантів ви отримаєте не тільки реальний досвід бізнес-аналізу, дизайну та впровадження інновацій, а й ряд фінансових та реалізаційних бонусів.
* Айкідо. Йога. Медитація yважності-yсвідомлення. Це не тільки спосіб відпочити чи розслабитись. Це інструмент покращення ефективності та досягнення емоційної стабільності через внутрішній особистий розвиток. За прикладом ІТ-флагманів, на кшталт Google чи Apple, Edvantis надає працівникам можливість торкатися до східних традицій та дисциплін, які покликані відтворювати внутрішній баланс та цілісність.

# 2. ПОСТАНОВКА ЗАВДАННЯ ДЛЯ ВИКОНАННЯ МАГІСТЕРСЬКОГО ДОСЛІДЖЕННЯ

Завдання отримана на базі практики збігається із завданням магістерської роботи, яке складається з наступних пунктів:

1) Реалізувати алгоритм apriori для пошуку асоційцій та запропонувати модифікації для засотування алгоритму для аналізу дорожньо-транспортних пригод.

2) Проектування архітектури системи для аналізу даних на основі алгоритму apriori.

3) Створення прототипу графічного інтерфейсу користувача для системи.

# 3. РЕЗУЛЬТАТИ ВИКОНАННЯ ЗАВДАННЯ ДЛЯ МАГІСТЕРСЬКОГО ДОСЛІДЖЕННЯ

## **3.1. Опис відомих модифікацій**

Проаналізувавши суміжні дослідження було виявлено що вже існує ряд модифікацій для алгоритму apriori, які покращують його роботу.

PredictiveApriori - клас реалізації інтелектуального алгоритму Apriori для видобування асоціативних правил. Він виконує пошук зі збільшенням порогу підтримки для кращих 'N' правил, що стосуються скоригованого значення достовірності на основі підтримки [1].

Правило додається, якщо: очікувана прогностична точність цього правила є одним з 'N' найкращих і не поглинена правилом, принаймні тієї ж очікуваної точності прогнозу. Якщо включені асоціативні правила видобуваються замість (загальних) асоціативних правил.

Якщо встановлено значення -1, останній атрибут буде взятий в якості атрибута класу та існує кількість правил для знаходження. Можливі пропущені значення, бінарні атрибути, порожні номінальні атрибути та номінальний клас.

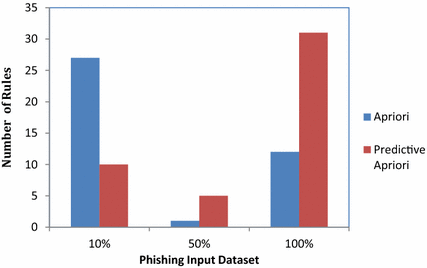


Рис. 3.1. Порівняльна харектеристика роботи apriori та модифікованого PredictiveApriori

Алгоритм AprioriTid[2] є різновидом алгоритму Apriori. Відмінною рисою даного алгоритму є підрахунок значення підтримки кандидатів не при скануванні безлічі D, а за допомогою безлічі Ck, які є множиною кандидатів (k- елементних наборів) потенційно частих, у відповідність яким ставиться ідентифікатор TID транзакцій, в яких вони містяться.

Кожен член безлічі Ck є парою виду , де кожен Fk є потенційно частим k - елементним набором, представленим в транзакції з ідентифікатором TID. Безліч C1 = D відповідає безлічі транзакцій, хоча кожен об'єкт в транзакції відповідає однооб'єктному набору в безлічі C1, що містить цей об'єкт. Для k> 1 безліч Ck генерується відповідно до алгоритму, описаним нижче.

Член безлічі Ck, відповідний транзакції Т, є парою такого вигляду:



Підмножина наборів в Ck з однаковими TID (містяться в одній і тій же транзакції) називається записом. Якщо транзакція не містить ні одного k - елементного кандидата, то Ck не матиме записи для цієї транзакції Тобто кількість записів в Ck може бути менше, ніж в D, особливо для великих значень k. Крім того, для великих значень k кожна запис може бути менше, ніж відповідна їй транзакція, т. К. В транзакції буде міститися мало кандидатів. Однак для малих знань k кожен запис може бути більше, ніж відповідна транзакція, тобто Ck включає всіх кандидатів k-елементних наборів, що містяться в транзакції.

Апріорі-TID використовує генерацію кандидата і апріорну функцію для визначення кандидата для наборів перед початком проходу. Основна відмінність від Apriori в тому, що він не використовує базу даних, для підрахунку підтримки після першого проходу. Швидше, він використовує 41 кодування кандидата наборів елементів, який використовується в попередному прохододі і позначається Ck. У Apriori-TID, кандидат в Ck наборів зберігаються в масиві індексується TID в Ck наборів. Кожен Ck зберігається в послідовної структури. У проході k-го, Апріорі-TID потребує обсяг пам'яті для Lk-1 і Ck при генерації кандидатів. Було також встановлено, що Апріорі-TID Перевищує Апріорі, коли є менше число наборів Ck, яке може поміститися в пам'яті і розподіл з великих наборів має довгий набір . Це означає, що розподіл записів у великих наборів високий на ранній стадії [3].

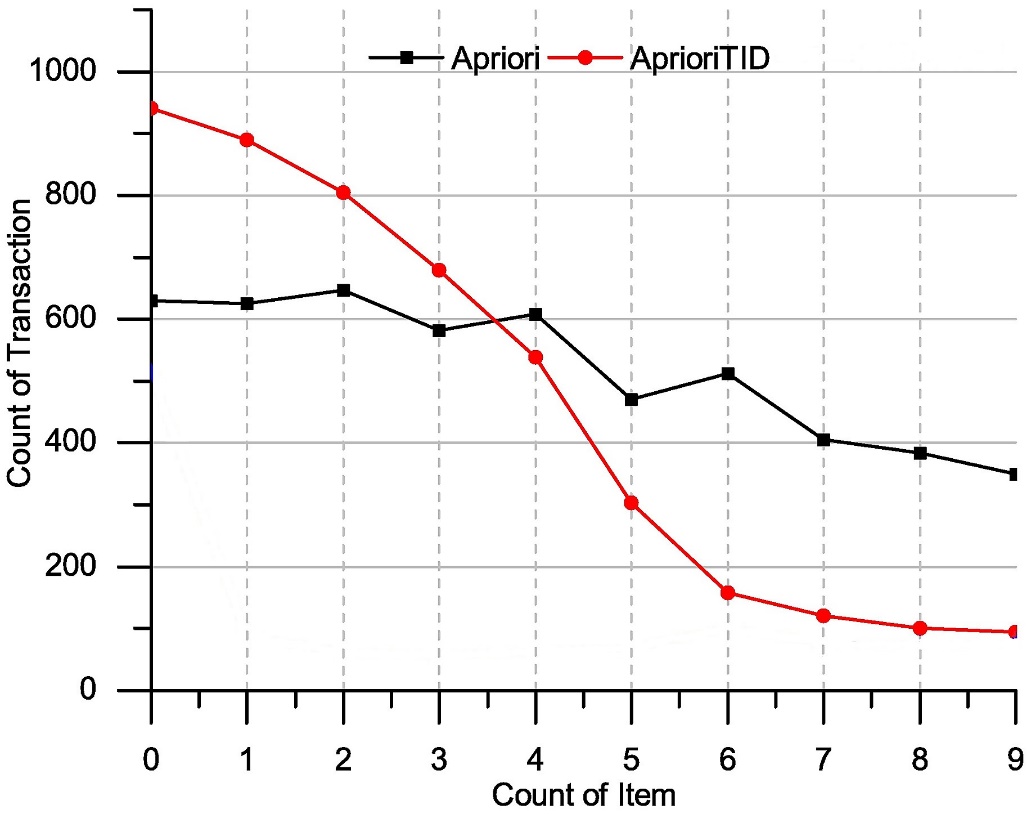


Рис. 3.2. Порівняльна харектеристика роботи apriori та модифікованого AprioriTID

Іншим різновидом алгоритму Apriori є алгоритм MSAP (Mining Sequential Alarm Patterns), спеціально розроблений для виконання секвенціального аналізу збоїв телекомунікаційної мережі. Він використовує наступне властивість підтримки послідовностей: для будь-якій послідовності Lk її підтримка буде менше, ніж підтримка послідовностей з безлічі Lk1. Алгоритм MSAP для пошуку подій, що слідують один за одним, використовує поняття "термінового вікна" (Urgent Window). Це дозволяє виявляти не просто однакові послідовності подій, а наступні один за одним. В іншому даний алгоритм працює за тим же принципом, що і Apriori[4].

## **3.2. Алгоритм apriori**

Алгоритм Apriori [4] призначений для знаходження всіх частих наборів елементів множини транзакцій, що аналізується. Він працює порівнево, використовуючи стратегію пошуку в ширину і виконується зверху вниз. Алгоритм Apriori має просту структуру, завдяки якій в неї можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних для вирішення задачі пошуку асоціативних правил.

Ключовим поняттям в алгоритмі Апріорі є антимонотонність міри підтримки. Це передбачає що:

* Всі підмножини частої Набір повинні бути частими
* Подібним чином, для будь-якого незвичного набору, всі його суперсеті також

повинні бути нечастими.

Cкладність цього алгоритму залежить від мінімального значення підтримки. Припустимо, що кількість вхідних транзакцій N, а кількість унікальних елементів R. Складність для генерації набору розміру i – О(Ri), де і – час для обчислення підтримки для кожного набору може бути O(N).

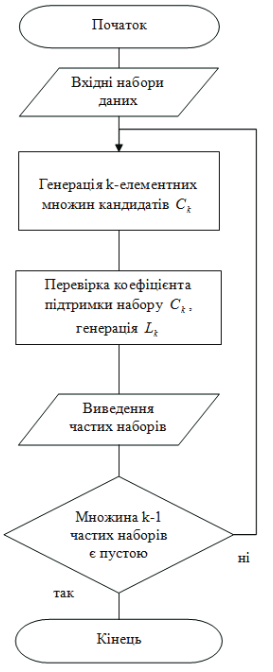


Рис. 3.3. Приклад роботи алгоритму у вигляді блок схеми

## **3.3. Опис модифікацій алгоритму для аналізу дорожньо-транспорних пригод**

Отже, переглянувши декілька відомих модифікацій, проаналізувавши предметну область та зрозумівши принцип роботи алгоритму та пошуку асоціативних правил було запропоновано модифікації для алгоритму.

Відомо, що алгоритм на кожному кроці визначає:

* підтримку (англ. support) - скільки разів у всьому масиві використано елементи даних, що складаються з X та Y
* достовірність (англ. confidence) - який відсоток від всіх одиниць, що містять X, містить також і Y

Асоціативні правила мають також характеристику під назвою Ліфт (англ. lift) – це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому. Чим більше значення ліфта, тим частіше наслідок визначається умовою в порівнянні з випадками, коли умова відсутня. Якщо ліфт дорівнює 1, зв'язок відсутній, близькі ж до нуля значення свідчать про сильну зворотну залежність.

Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсіканання абсурдних або випадкових груп правил, оттже швидкість алгоритму повинна зрости адже не потрібно здійснюватинаступні кроки по вже відкинутих правилах.

Ще одним із можливих варіантів модифікації є розбиття всієї множини всіх можливих характеристик ДТП на дві підмножини:

1) Si - характеристики, сукупність яких могла стати причиною виникнення ДТП;

2) Di - характеристики, сукупність яких описує ДТП що виникло.

Асоціативні правила, що містять лише підмножину характеристик із множини Si, не містять корисної інформації для виконання подальшого аналізу, завдяки чому можна прискорити алгоритм пошуку асоціативних правил. Для цього необхідно змінити процес генерації наборів кандидатів Ck , в якому відсікати набори, що не містять характеристик із множини Di. В результаті значно зменшиться кількість наборів кандидатів Ck , що генеруватиметься на кожній ітерації алгоритму, а також це впливатиме на генерацію частих наборів Lk. Час обробки множини наборів Ck та генерації Lk лінійно залежить від потужності даних множин і загальної потужності множини І, яка є сталою в процесі роботи алгоритму. Тому завдяки зменшенню кількості елементів множини Ck зменшиться загальний час роботи алгоритму.

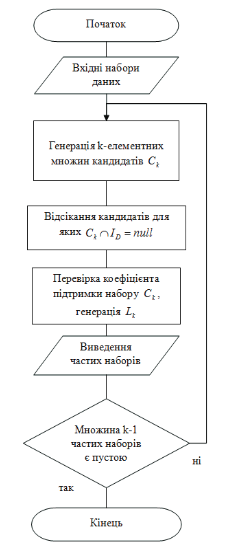


Рис. 3.4. Приклад роботи модифікованого алгоритму у вигляді блок схеми

## **3.4. Прототипування графічного інтерфейсу користувача**

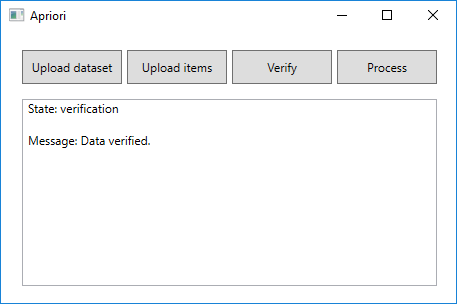


Рис 3.5. Головне меню програми

На головному меню користувачеві повинна бути надана можливість завантажити увесь «датасет» та предмети які є у цій базі. Наступний крок для користувача це перевірка чи збігаються предмети із базою і навпаки.

Також користувач має можливість почати процес знаходження асоціативних правил натиснувши кнопку «Process».

На кожному етапі виводиться повідовлення про результат виконання.

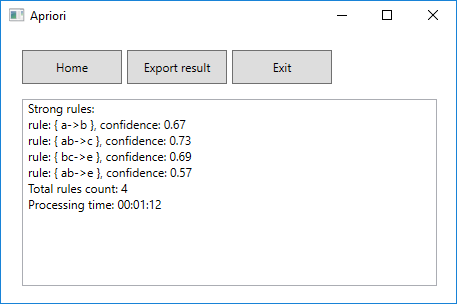


Рис 3.6. Результати виконання програми

Робоча область програми являє собою вікно на якому предтавлено результати виконання програми.

Також користувач повинен мати можливість згенерувати звіт про проведення аналізу.

## **3.5.** **Проектування архітектури системи для роботи алгоритму apriori**



Рисунок 3.7. UML діаграма класів програми

Створення програми відбувається згідно з розробленою діаграмою класів програми. Розглянемо складові діаграми, щоб детальніше зрозуміти роботу програми.

Клас FileReader відповідає за зчитування даних із файлів та конвертання їх у масив. Після відкриття файлу з’являється діалогове вікно для вибору .txt файлу.

Після зчитування файлу, клас Verification відповідає за перевірку даних із предметам і навпаки. Клас Sorter сортує елементи в правилах за зростанням по назвах, щоб уникати повторень. Клас Item містить інформацію про кожен предмет який відповідає вравилам. Rule містить інформацію про правило X – лівий операнд, Y правий операнд, та Confidence – вірогідність. Клас Output – містить інформацію про результат роботи алгоритму.

Основна логіка виконується а класі Apriori – клас який асимілює всі перелічені вище класи та має публічний метод ProcessTransaction, який на вхід отримує вхідні дані: масив «items», масив «dataSet», мінімальне значення вірогідності та підтримки. Та після обробки повертає результат типу Output.

# 4. ВИСНОВКИ

Після аналізу існуючих систем, що займаються пошуком закономірностей у даних, огляду існуючих підходів та алгоритмів для опрацювання великих даних було виявлено потребу у дослідженні роботи алгоритму для напіструктурованих даних, так як досліджень аналогів не було виявлено та виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Здійснено обгрунтування та порівняльний аналіз алгоритму apriori, визначено що алгоритм є простий у розумінні та схильний до модифікацій.

Запропоновано покращення базового алгоритму, за допомогою застосування характиристики Ліфту на кожному кроці алгоритму. Даний підхід повинен зменшити витрати часу при знаходженні асоціативних правил. Також є можливим впровадження даного підходу до уже відомих модифікацій, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Також запропоновано ще один варіант у якому вхідна множина розділяється на дві підмножини, перша множина містить характеристики сукупність яких согла стати причиною виникненн ДТП, і друга множина -характеристики сукупність яких описує ДТП що виникло, на кожному кроці відсікається перша множина, адже вона не містить корисної інформаці.

Під час проходження практики мною було покращено навички та вміння у роботі із технологією .Net. Також я покращив свої навички стосовно застосування шаблонів проектування та архітектурних рішень, рефакторингу коду. Покращено знання і навички роботи з методологією розробки SCRUM на практиці.

Було розроблено прототип системи для магістерської роботи та підготовлено матеріали для ІV міжнародної науково-технічної конференція «Комп'ютерне моделювання та оптимізація складних систем» м.Дніпро.

# 5. СПИСОК ОПРАЦЬОВАНИХ ДЖЕРЕЛ

1. Laney, Doug. "3D data management: Controlling data volume, velocity and variety". META Group Research Note. (09.09.2018) 45–60.
2. Marr, Bernard "Big Data: The 5 Vs Everyone Must Know". (09.09.2018). 98–103
3. Пошук асоціативних правил для аналізу надзвичайних ситуацій на залізничному транспорті [Веб-сайт]. – Електронні дані. – Режим доступу:

https://itce.vntu.edu.ua/index.php/itce/article/view/27/28 (дата звернення: 09.05.2018)

1. Mohammed J. Zaki. Scalable algorithms for association mining / Mohammed J. Zaki // IEEE Transactions on Knowledge and Data Engineering, 12 (3): 372-390, May/June 2000.
2. Brin S. Dynamic itemset counting and implication rules for market basket data / Brin S., Motwani R., Ullman J., Tsur S // SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, pages 255-264, Tucson, Arizona, USA, May 1997.
3. WebApi [Електронний ресурс] : – Режим доступу: <https://www.asp.net/web-api>
4. Дюк В., Самойленко А. Data Mining: Учебный курс. – СПб: Питер, 2001. – 368 с.
5. А. Шахиди. Data Mining – добыча данных [Електронний ресурс] // Big Data: сайт. - Режим доступу // <http://www.basegroup.ru>
6. Karen Montgomery. Big Data Now: 2014 Edition. O'Reilly Media.- Junuary, 2015 – 165p.
7. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. John Wiley & Sons. 2014-12-19. 300p.
8. Apache Hadoop. [Електронний ресурс] // Big Data: сайт. - Режим доступу https://hadoop.apache.org/docs/r1.2.1/hdfs\_design.html
9. Tong Qiang, Zhou Yuanchun, Wu Kaichao, Yan Baoping. A quantitative association rules mining algorithm[J]. Computer engineering. 2007

# Додаток A. Код РЕАЛІЗАЦІЇ АЛГОРИТМУ

using System.Linq;

using System.Text;

namespace Apriori.Implementation

{

public class Apriori : IApriori

{

readonly ISorter \_sorter;

public Apriori()

{

\_sorter = new Sorter();

}

Output IApriori.ProcessTransaction(double minSupport, double minConfidence, IEnumerable<string> items, string[] transactions)

{

IList<Item> frequentItems = GetL1FrequentItems(minSupport, items, transactions);

ItemsDictionary allFrequentItems = new ItemsDictionary();

allFrequentItems.ConcatItems(frequentItems);

IDictionary<string, double> candidates = new Dictionary<string, double>();

double transactionsCount = transactions.Count();

do

{

candidates = GenerateCandidates(frequentItems, transactions);

frequentItems = GetFrequentItems(candidates, minSupport, transactionsCount);

allFrequentItems.ConcatItems(frequentItems);

}

while (candidates.Count != 0);

HashSet<Rule> rules = GenerateRules(allFrequentItems);

IList<Rule> strongRules = GetStrongRules(minConfidence, rules, allFrequentItems);

Dictionary<string, Dictionary<string, double>> closedItemSets = GetClosedItemSets(allFrequentItems);

IList<string> maximalItemSets = GetMaximalItemSets(closedItemSets);

return new Output

{

StrongRules = strongRules,

MaximalItemSets = maximalItemSets,

ClosedItemSets = closedItemSets,

FrequentItems = allFrequentItems

};

}

private List<Item> GetL1FrequentItems(double minSupport, IEnumerable<string> items, IEnumerable<string> transactions)

{

var frequentItemsL1 = new List<Item>();

double transactionsCount = transactions.Count();

foreach (var item in items)

{

double support = GetSupport(item, transactions);

if (support / transactionsCount >= minSupport)

{

frequentItemsL1.Add(new Item { Name = item, Support = support });

}

}

frequentItemsL1.Sort();

return frequentItemsL1;

}

private double GetSupport(string generatedCandidate, IEnumerable<string> transactionsList)

{

double support = 0;

foreach (string transaction in transactionsList)

{

if (CheckIsSubset(generatedCandidate, transaction))

{

support++;

}

}

return support;

}

private bool CheckIsSubset(string child, string parent)

{

foreach (char c in child)

{

if (!parent.Contains(c))

{

return false;

}

}

return true;

}

private Dictionary<string, double> GenerateCandidates(IList<Item> frequentItems, IEnumerable<string> transactions)

{

Dictionary<string, double> candidates = new Dictionary<string, double>();

for (int i = 0; i < frequentItems.Count - 1; i++)

{

string firstItem = \_sorter.Sort(frequentItems[i].Name);

for (int j = i + 1; j < frequentItems.Count; j++)

{

string secondItem = \_sorter.Sort(frequentItems[j].Name);

string generatedCandidate = GenerateCandidate(firstItem, secondItem);

if (generatedCandidate != string.Empty)

{

double support = GetSupport(generatedCandidate, transactions);

candidates.Add(generatedCandidate, support);

}

}

}

return candidates;

}

private string GenerateCandidate(string firstItem, string secondItem)

{

int length = firstItem.Length;

if (length == 1)

{

return firstItem + secondItem;

}

else

{

string firstSubString = firstItem.Substring(0, length - 1);

string secondSubString = secondItem.Substring(0, length - 1);

if (firstSubString == secondSubString)

{

return firstItem + secondItem[length - 1];

}

return string.Empty;

}

}

private List<Item> GetFrequentItems(IDictionary<string, double> candidates, double minSupport, double transactionsCount)

{

var frequentItems = new List<Item>();

foreach (var item in candidates)

{

if (item.Value / transactionsCount >= minSupport)

{

frequentItems.Add(new Item { Name = item.Key, Support = item.Value });

}

}

return frequentItems;

}

private Dictionary<string, Dictionary<string, double>> GetClosedItemSets(ItemsDictionary allFrequentItems)

{

var closedItemSets = new Dictionary<string, Dictionary<string, double>>();

int i = 0;

foreach (var item in allFrequentItems)

{

Dictionary<string, double> parents = GetItemParents(item.Name, ++i, allFrequentItems);

if (CheckIsClosed(item.Name, parents, allFrequentItems))

{

closedItemSets.Add(item.Name, parents);

}

}

return closedItemSets;

}

private Dictionary<string, double> GetItemParents(string child, int index, ItemsDictionary allFrequentItems)

{

var parents = new Dictionary<string, double>();

for (int j = index; j < allFrequentItems.Count; j++)

{

string parent = allFrequentItems[j].Name;

if (parent.Length == child.Length + 1)

{

if (CheckIsSubset(child, parent))

{

parents.Add(parent, allFrequentItems[parent].Support);

}

}

}

return parents;

}

private bool CheckIsClosed(string child, Dictionary<string, double> parents, ItemsDictionary allFrequentItems)

{

foreach (string parent in parents.Keys)

{

if (allFrequentItems[child].Support == allFrequentItems[parent].Support)

{

return false;

}

}

return true;

}

private IList<string> GetMaximalItemSets(Dictionary<string, Dictionary<string, double>> closedItemSets)

{

var maximalItemSets = new List<string>();

foreach (var item in closedItemSets)

{

Dictionary<string, double> parents = item.Value;

if (parents.Count == 0)

{

maximalItemSets.Add(item.Key);

}

}

return maximalItemSets;

}

private HashSet<Rule> GenerateRules(ItemsDictionary allFrequentItems)

{

var rulesList = new HashSet<Rule>();

foreach (var item in allFrequentItems)

{

if (item.Name.Length <= 1) continue;

IEnumerable<string> subsetsList = GenerateSubsets(item.Name);

foreach (var subset in subsetsList)

{

string remaining = GetRemaining(subset, item.Name);

Rule rule = new Rule(subset, remaining, 0);

if (!rulesList.Contains(rule))

{

rulesList.Add(rule);

}

}

}

return rulesList;

}

private IEnumerable<string> GenerateSubsets(string item)

{

IEnumerable<string> allSubsets = new string[] { };

int subsetLength = item.Length / 2;

for (int i = 1; i <= subsetLength; i++)

{

IList<string> subsets = new List<string>();

GenerateSubsetsRecursive(item, i, new char[item.Length], subsets);

allSubsets = allSubsets.Concat(subsets);

}

return allSubsets;

}

private void GenerateSubsetsRecursive(string item, int subsetLength, char[] temp, IList<string> subsets, int q = 0, int r = 0)

{

if (q == subsetLength)

{

StringBuilder sb = new StringBuilder();

for (int i = 0; i < subsetLength; i++)

{

sb.Append(temp[i]);

}

subsets.Add(sb.ToString());

}

else

{

for (int i = r; i < item.Length; i++)

{

temp[q] = item[i];

GenerateSubsetsRecursive(item, subsetLength, temp, subsets, q + 1, i + 1);

}

}

}

}

}