# Анотація

Магістерська кваліфікаційна робота присвячена дослідженню та реалізації алгоритму Apriori, та введенню можливих модифікацій для кращої роботи алгоритму.

Проект включає програмну реалізацію алгоритму Apriori та проведення дослідження для цього алгоритму.

Проведено дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму для різних значень коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

Основна мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

Загальний обсяг роботи 76 сторінок.

# ABSTRACT

The master's qualification work is devoted to the research and development of Apriori algorithm and provide some modifications for better performance.

The project includes software implementation of the Apriori algorithm and conducting experiments of the developed tool.

The Research was conducted for basic algorithm without modifications, the study of the modified algorithm by the first method, the research of the modified algorithm by the second method, research of the algorithm's performance for different values ​​of the coefficients and a comparative analysis of the above-mentioned studies.

The main purpose of master's qualification work is to compare the work of the basic Apriori algorithm and its modifications on large volumes of data. It is necessary to process the data with the specified algorithm and estimate the speed of the algorithm, the load on the resources, the number of associative rules found on certain data sets. To work out on the basis of computational experiments a proposal for modification of the algorithm.

The object of the research is the apriori algorithm and its modifications.

Subject of research is the time and accuracy of the search for associative rules and the search for possible modifications.

The total workload is 76 pages.

# ВСТУП

На сьогоднішній день ІТ є однією з найбільш динамічніших сфер, активно розробляються нові технології, створюються сотні програмних продуктів та накопичаються приблизно декілька сотень петабайт інформації у день. Ця інформація надходить з різних джерел, та містить багато даних.

Big Data відноситься до надзвичайно великих масивів даних, які важко аналізувати традиційними інструментами. Найбільш відомим та широко застосовуваним зараз є алгоритм Apriori.

Алгоритм Apriori 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.

Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

У першому розділі даної роботи проведено аналіз предметної області. 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, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

У другому розділі цієї роботи подано опис алгоритму та можливі модифікації. Для першої модифікації потрібно розбити множину всіх можливих транспортних пригод на дві підмножини, одна з них містить корисну інформацію для аналізу, а інша ні, отже та що не містить корисної інформації мотрібно відкидати на кожному кроці виконання алгоритму. Для другої модифікації потрібно додати характеристику асоціативних правил Ліфт, та обчислювати його на кожному кроці алгоритму.

Третій розділ роботи містить специфікацію вимог до програмної реалізації алгоритму представленого у розділі 2, інформацію про використані технології, графічний інтерфейс програмного засобу та певні особливості програмної реалізації засобу для проведення аналізу даних алгоритмом Apriori.

У четвертому розділі показано дослідження розробленого засобу. Дослідження складається з таких частин: дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму при різними значеннями коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

У останньому розділі даної роботи показано економічну характеристику програмного продукту, розробленого у розділі 3.

# РОЗДІЛ 1. АНАЛІЗ предметної області та суміжних досліджень.

* 1. **Великі дані, визначення та їх особливості**

Великі дані - це набори даних, настільки об'ємні та складні, що традиційне прикладне програмне забезпечення для обробки даних є недостатнім для їх вирішення. Великі задачі даних включають в себе захоплення даних, зберігання даних, аналіз даних, пошук, обмін, передачу, візуалізацію, запити, оновлення, конфіденційність інформації та джерела даних. Існує ряд понять, пов'язаних з великими даними: спочатку було 3 поняття обсягу, різноманітності, швидкості[1]. Іншими поняттями, які потім пояснюються великими даними, є достовірність (тобто, скільки шуму в даних) [2] та значення. Останнім часом термін "великі дані" має на увазі використання прогнозної аналітики, аналізу поведінки користувачів або деяких інших розширених методів аналізу даних, які витягують значення з даних, а рідко - до певного розміру набору даних. "Існує мало сумнівів у тому, що кількість даних, які зараз доступні, дійсно велика, але це не є найважливішою характеристикою цієї нової екосистеми даних". Аналіз наборів даних може знайти нові зв'язки з "точковими діловими тенденціями, запобігання захворюванням, боротьба з злочинністю та ін. "[3]

Вчені, керівники підприємств, практики медицини, реклами та уряди так само регулярно зустрічаються з труднощами з великими наборами даних у таких областях, як пошук в Інтернеті, технології, міська інформатика та бізнес-інформатика. Вчені стикаються з обмеженнями в роботі електронної науки, включаючи метеорологію, геноміку, сонеономію, складні фізичні симуляції, біологію та екологічні дослідження [4].

Цей термін використовується з 1990-х років, при цьому деякі приносять кредит Джону Маші за виготовлення або, принаймні, популярністю. [5] Великі дані зазвичай включають в себе набори даних, розміри яких не залежать від можливостей широко використовуваних програмних інструментів для захоплення, копіювання, керування та обробки даних протягом допустимого минулого часу [6]. Велика філософія даних охоплює неструктуровані, напівструктуровані та структуровані дані, однак основна увага приділяється неструктурованим даним.

Визначення 2016 р. Свідчить, що "великі дані являють собою інформаційні активи, які характеризуються такими великими обсягами, швидкістю та різноманітністю, що вимагають спеціальних технологій та аналітичних методів перетворення їх у вартість" [7].

Великі дані можна описати за такими характеристиками: [8]

**Обсяг** (англ. Volume) **–** кількість створених та збережених даних. Розмір даних визначає цінність та потенційне розуміння, і чи можна вважати великі дані чи ні.

**Різноманітність** (англ. Variety) **–** тип та характер даних. Це допомагає людям, які аналізують його, ефективно використовувати отримане розуміння. Великі дані витягуються з тексту, зображень, аудіо, відео; плюс він завершує відсутність частин через злиття даних.

**Швидкість** (англ. Velocity) **–** у цьому контексті швидкість, з якою дані створюються та обробляються, щоб задовольнити вимоги та виклики, які лежать на шляху зростання та розвитку. Великі дані часто доступні в режимі реального часу.

Дані треба обробляти за допомогою розширених інструментів (аналітики та алгоритмів) для виявлення значущої інформації. Наприклад, щоб керувати фабрикою, слід розглядати як видимі, так і невидимі проблеми з різними компонентами. Алгоритми формування інформації повинні виявляти та вирішувати невидимі проблеми, такі як деградація машини, знос компонентів тощо на заводі. [9]

* 1. **Поняття асоціативних правил**

Афінітивний аналіз (affinity analysis) — один з розповсюджених методів Data Mining. Його назва походить від англійського слова affinity, що у перекладі означає «близькість», «подібність». Ціль даного методу — дослідження взаємного зв'язку між подіями, які відбуваються спільно. Різновидом афінітивного аналізу є аналіз ринкового кошика (market basket analysis), ціль якого - виявити асоціації між різними подіями, тобто знайти правила для кількісного опису взаємного зв'язку між двома або більше подіями. Такі правила називаються асоціативними правилами (association rules).[10]

Базовим поняттям у теорії асоціативних правил є транзакція — деяка множина подій, що відбуваються спільно. Типова транзакція - покупка клієнтом товару в супермаркеті. У переважній більшості випадків клієнт купує не один товар, а набір товарів, що називається ринковим кошиком. При цьому виникає питання: чи є покупка одного товару в кошику наслідком або причиною покупки іншого товару, тобто, чи пов'язані дані події? Цей зв'язок і встановлюють асоціативні правила. Наприклад, може бути виявлене асоціативне правило, котре стверджує, що клієнт, який купив молоко, з імовірністю 75 % купить і хліб.

Наступне важливе поняття — предметний набір. Це непорожня множина предметів, що з'явилися в одній транзакції.

Аналіз ринкового кошика - це аналіз наборів даних для певної комбінації товарів, пов'язаних між собою. Іншими словами, виконується пошук товарів, присутність яких у транзакції впливає на ймовірність наявності інших товарів або комбінацій товарів.

Сучасні касові апарати в супермаркетах дозволяють збирати інформацію про покупки, що може зберігатися в базі даних. Потім накопичені дані можуть використовуватися для побудови систем пошуку асоціативних правил.

У табл. 1.1 представлений простий приклад, що містить дані про ринковий кошик. У кожному рядку вказується комбінація продуктів, придбаних за одну покупку. Хоча на практиці доводиться мати справу з мільйонами транзакцій, у яких беруть участь десятки й сотні різних продуктів, приклад обмежений 10 транзакціями, що містять 13 видів продуктів: аби проілюструвати методику виявлення асоціативних правил, цього досить.

Таблиця 1.1.

Приклад набору транзакцій

|  |  |
| --- | --- |
| **№** | **Транзакція** |
| 1 | Сливи, салат, помідори |
| 2 | Селера, цукерки |
| 3 | Цукерки |
| 4 | Яблука, морква, помідори, картопля, цукерки |
| 5 | Яблука, апельсини, салат, цукерки, помідори |
| 6 | Персики, апельсини, селера, помідори |
| 7 | Квасоля, салат, помідори |
| 8 | Апельсини, салат, морква, помідори, цукерки |
| 9 | Яблука, банани, сливи, морква, помідори, цибуля, цукерки |
| 10 | Яблука, картопля |

Візуальний аналіз прикладу показує, що всі чотири транзакції, у яких фігурує салат, також включають помідори, і що чотири із семи покупок, що містять помідори, також містять салат. Салат і помідори в більшості випадків купуються разом. Асоціативні правила дозволяють виявляти й кількісно описувати такі збіги.

Асоціативне правило складається із двох наборів предметів, що мають назву умова (antecedent) та наслідок (consequent), й записуються у вигляді . Таким чином, асоціативне правило формулюється у вигляді: «Якщо умова, то наслідок».

Умова може обмежуватися тільки одним предметом. Правила звичайно відображаються за допомогою стрілок, спрямованих від умови до наслідку, наприклад, помідори → салат.

Візуальний аналіз прикладу показує, що всі чотири транзакції, у яких фігурує цибулю, також включають картоплю, і що чотири із семи шести, що містять картоплю, також містять цибулю. Цибулю і картоплю в більшості випадків купуються разом. Асоціативні правила дозволяють виявляти й кількісно описувати такі збіги. Також правила можуть бути комбіновані {Цибуля, Картопля} => {Бургер}

Умова може обмежуватися тільки одним предметом. Правила звичайно відображаються за допомогою стрілок, спрямованих від умови до наслідку, наприклад, цибуля → картопля.

Основними характеристиками, що описують асоціативне правило, є підтримка (support) і вірогідність (confidence)[29-31].

Якщо позначити базу даних транзакцій через , а число транзакцій у цій базі , то кожна транзакція  являє собою певний набір предметів. Позначимо підтримку правила через , а вірогідність – через .

Підтримка асоціативного правила — це число транзакцій, які містять як умову, так і наслідок. Наприклад, для асоціації  можна записати[30]

 (1.1)

Вірогідність асоціативного правила  являє собою міру точності правила й визначається як відношення кількості транзакцій, що містять умову і наслідок, до кількості транзакцій, що містять тільки умову:

 (1.2)

Якщо підтримка й вірогідність досить високі, можна з великою ймовірністю стверджувати, що будь-яка майбутня транзакції, що включає умову, буде містити й наслідок.

Для прикладу розглянемо правило салат → помідори, що напрошується з попередніх спостережень . Для нього

;

.

Дане правило, що зустрічається в 40% транзакцій, є для вихідних даних абсолютно вірним – у всіх випадках, коли клієнт купує салат, він разом з тим купує й помідори. Це легко пояснити логічно - обидва продукти використовуються для готування овочевих блюд і дійсно часто купуються разом.

Тепер розглянемо асоціацію , у якій містяться слабко сумісні в гастрономічному плані продукти.

*помідори*

*цукерки*



Підтримка даної асоціації S = 4/10 = 0,4 – та ж, що в попереднього правила, а вірогідність С = 4/6 = 0,67. Таким чином, порівняно невисока вірогідність даної асоціації дає привід засумніватися в тому, що вона є правилом.

Число 0,67 здається не таким вже й малим. Чому ж ми говоримо про «незначну вірогідність» цієї асоціації? Справа в тому, що помідори зустрічаються в 7 чеках з 10 ( ). Прийнято, що всі правила з вірогідністю меншою, ніж проста ймовірність випадання наслідку не повинні розглядатися, адже вони, по суті, випадкові. Для прийняття асоціативного правила його вірогідність повинна бути не меншою ніж імовірність наслідку.

Останнє зауваження при досить великій номенклатурі товарів призводить до необхідності подвоювати кількість розрахунків. На практиці аналітики можуть віддавати перевагу правилам, які мають високу підтримку (вище певного рівня, наприклад, 0,3) або високу вірогідність (не менше 0,8-0,85). Висування одночасних вимог щодо підтримки й вірогідності дозволяють значно позм'якшити критерії (підтримку до 0,1-0,15, вірогідність – до 0,67-0,75). Правила, для яких значення підтримки й вірогідності перевищують певні, задані користувачем пороги, називають сильними правилами (strong rules). Всі наведені вище числові значення - емпіричні. Наприклад, у задачах виявлення шахрайських операцій значення підтримки може знижуватися й до 1%, оскільки із шахрайством пов'язане порівняно невелике число транзакцій.

Крім об'єктивних оцінок (підтримки й вірогідності) кожного зі згенерованих правил, різні джерела радять використовувати деякі суб'єктивні оцінки. Всі вони, так чи інакше, базуються на об'єктивних.

Ліфт (від interest lift – підвищення інтересу) обчислюється в такий спосіб

(1.3)

Ліфт - 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[28].

Для нашого прикладу в таблиці 1.1. візьмемо два правила з однаковою вірогідністю:

 4/10 = 0,4. ,4/7 = 0,57.

*салат) =*

*C(помідори*



 6/10 = 0,6. 4/7 = 0,57.

*цукерки) =*

*C(помідори*



Здавалося б, правила однаково достовірні. Після розрахунку ліфта все стає на свої місця:

0,57/0,4 = 1,425;-

*салат) =*

*L(помідори*



0,57/0,6 = 0,95.

*цукерки) =*

*L(помідори*



Не варто вважати ліфт універсальним мірилом адекватності. Справа в тому, що правило з меншою підтримкою й більшим ліфтом може бути менш значимим, ніж альтернативне правило з більшою підтримкою й меншим ліфтом. Це пов'язане з тим, що останнє застосовується для більшого числа покупців.

Левередж (leverage – важіль, плече) – це різниця між спостережуваною частотою, з якої умова й наслідок з'являються спільно, та добутком частот появи умови й наслідку окремо

 (1.4)

Левередж дозволяє впоратися із ситуаціями, коли й підтримка, й ліфт у правил ідентичні, але їх значимість явно відрізняється. Наприклад, у нашому овочевому магазині в правила 

;  0,7;  1/0,7 = 1,43.

Морква, як показує таблиця 1.1, також продається тільки з помідорами, і також зустрічається чотири рази, тому й у правила 

;  0,7;  1/0,7 = 1,43.

А от левередж у цих правил відрізняється на 30%:

 0,3 – 0,3 \* 0,7 = 0,09.

 0,4 – 0,4 \* 0,7 = 0,12.

Таким чином, значимість другої асоціації більша, ніж першої.

Альтернативою левередж є поліпшення.

Поліпшення (improvement) – це відношення частоти спостережуваних виконань правила до добутку частот появи умови й наслідку окремо.

 (1.5)

фактично, поліпшення показує, у скільки разів розглянуте правило забезпечує правильний прогноз краще, ніж випадкове вгадування. Всі правила  не є значимими.

Такі міри, як ліфт, левередж і поліпшення, можуть використовуватися для обмеження набору розглянутих асоціацій шляхом встановлення граничних значень значимості, нижче яких асоціації відкидаються.

* 1. **Аналіз існуючих досліджень**

На сьогоднішній день існує багато досліджень та для алгоритму Apriori.

У [11] проаналізовано алгоритм Apriori як метод пошуку асоціативних правил у структурованих та неструктурованих даних з погляду кількості знайдених правил, швидкодії та потреб в обчислювальних ресурсах. Неструктуровані дані тісно пов’язані з терміном Big Data. Актуальним завданням інженерії даних є виявлення ефективних засобів опрацювання неструктурованої інформації. Для проведення обчислювальних експериментів розроблено програмну систему, що опрацьовує дані алгоритмом Apriori, предметною областю якої вибрано торгівлю. Така система може бути прототипом реальної рекомендаційної системи.

Отже було досліджено що,

1. Втрати у швидкодії для неструктурованої інформації порівняно зі структурованою інформацією незначні зі зростанням обсягів інформації, і зокрема, для випадку обсягу даних 2,5 Гб зростання часу роботи становить близько 12 %. А на порівняно невеликих обсягах даних для структурованих даних алгоритм працює утричі швидше порівняно із неструктурованими.

2. Кількість знайдених правил для неструктурованої інформації на близько 25 % менша, ніж для структурованої інформації.

3. Для коефіцієнтів підтримки та достовірності, вищих від середнього, для неструктурованих даних не знайдено жодного правила, а для структурованих результат – від 2 до 5 правил.

У [12] виконувався аналіз надзвичайних ситуацій у залізничному транспорті для пошуку асоціативних правил із використанням алгоритму Apriori. У цьому дослідженні запропоновано модифікацію алгоритму: розбитя множини всіх можливих характеристик на дві підмножини:

1. A – характеристики, сукупність яких могла стати причиною виникнення надзвичайних ситуацій.
2. B – характеристики, скупність яких описує надзвичайну ситуацію що виникла.

В результаті застосування цієї модифікації збільшується швидкість роботи

алгоритму.

З отриманих результатів дослідження видно що алгоритм є доволі дослідженим та схильном до модифікацій проте, його можна досліджувати для навіструктурованих даних або ввести нові модифікації для більш точної обробки інформації.

* 1. **Мета, об’єкт та предмет дослідження**

Мета роботи полягає в порівняльному аналізі роботи алгоритму Apriori. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних. Виробити на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Об’єкт дослідження – алгоритм apriori та його модифікації.

Предмет дослідження – час і точність пошуку асоціативних правил та пошук можливих модифікацій.

* 1. **Висновок до розділу 1**

Накопичення інформації є актуальною темою сьогодення. Для аналізу цієї інформації використовують різноманітні технології та алгоритму. Після огляду декількох із них було вирішено провести дослідження для алгоритму Apriori.

Отже, у магістерській кваліфікаційній роботі необхідно розглянути алгоритм Apriori з точки зору пошуку асоціативних правил у великій кількості даних та проаналізувати затрачені обчислювальні ресурси. Мета роботи полягає в порівняльному аналізі роботи класичного алгоритму Apriori та його модифікацій. Необхідно опрацювати дані вказаним алгоритмом та оцінити швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил на певних наборах даних.

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, та на основі обчислювальних експериментів надати пропозицію для модифікації алгоритму.

# РОЗДІЛ 2. Опис алгоритму apriori

1. 1. **Алгоритм apriori**

На даний момент існує декілька основних алгоритмів пошуку асоціативних правил в загальному вигляді, що не зав’язані на конкретизовану предметну область. Розглянемо найбільш поширені з них.

Алгоритм Apriori [14] 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.

Алгоритм Eclat [15] побудований на основі пошуку в глибину використовуючи перетин наборів елементів для знаходження частих наборів елементів. В загальному випадку алгоритм працює повільніше ніж алгоритм Apriori, однак в ситуації коли кількість різних елементів транзакцій є невеликою, він працює швидше.

Алгоритм FP-growth [16] (frequent pattern growth, зростання частих патернів) використовує розширене префіксне дерево для збереження бази даних у стислому вигляді. Він застосовує метод «розділяй і володарюй» для декомпозиції і видобутку знань із бази даних. В процесі роботи алгоритм використовує метод зростаючих патернів для уникнення ресурсоємкого процесу генерації частих кандидатів і їх тестування, що використовується в алгоритмі Apriori. Однак через це ускладнюється процес оптимізації алгоритму для використання з специфічними наборами даних, де покращення швидкості роботи можна досягти іншими методами.

Отже, враховуючи переваги та недоліки розглянутих алгоритмів пошуку асоціативних правил, було обрано алгоритм Apriori так як він є простим у розумінні та можна легко вносити модифікації для оптимізації алгоритму під конкретні набори даних.[23-29]

Таблиця 2.1.

Приклад набору транзакцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction ID** | **Цибуля** | **Картопля** | **Бургер** | **Молоко** | **Пиво** |
| t_1 | 1 | 1 | 1 | 0 | 0 |
| t_2 | 0 | 1 | 1 | 1 | 0 |
| t_3 | 0 | 0 | 0 | 1 | 1 |
| t_4 | 1 | 1 | 0 | 1 | 0 |
| t_5 | 1 | 1 | 1 | 0 | 1 |
| t_6 | 1 | 1 | 1 | 1 | 1 |

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

Давайте розглянемо інтуїтивно зрозуміле пояснення алгоритму за допомогою наведеного вище прикладу(табл. 2). Перед початком процесу, встановіть порогової підтримки до 50%, тобто лише ті елементи значні, для яких підтримка більше 50%.

**Крок 1:** Створити таблицю частот для всіх елементів, що виникають у всіх транзакціях. Для нашого випадку:

Таблиця 2.2.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |
| Пиво(Be) | 2 |

**Крок 2:** Ми знаємо, що важливі лише ті елементи, для яких підтримка більше або дорівнює пороговій підтримці. Тут границя підтримки становить 50%, отже, лише ті елементи значущі, що трапляються у більш ніж трьох транзакціях, а такими є - Цибуля (O), Картопля (P), Бургер (B) і Молоко (M). Тому ми отримуємо наступну таблицю:

Таблиця 2.3.

Таблиця частот

|  |  |
| --- | --- |
| **Одиниця** | **Частота** |
| Цибуля(O) | 4 |
| Картопля(P) | 5 |
| Бургер(B) | 4 |
| Молоко(M) | 4 |

**Крок 3:** Наступним кроком є зробити всі можливі пари значущих елементів, маючи на увазі, що порядок не має значення, тобто АB такий самий, як BА. Щоб зробити це, візьміть перший предмет і з'єднайте його з усіма іншими, такими як OP, OB, OM. Аналогічним чином розглянемо другий елемент і об'єднає його з попередніми пунктами, тобто PB, PM. Ми розглядаємо лише попередні пункти, оскільки PO (як OP) вже існує. Отже, всі пари в нашому прикладі - OP, OB, OM, PB, PM, BM.

**Крок 4:** Тепер ми розглянемо випадки кожної пари у всіх транзакціях.

Таблиця 2.4.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
|  |  |
| OP | 4 |
| OB | 3 |
| OM | 2 |
| PB | 4 |
| PM | 3 |
| BM | 2 |

**Крок 5.** Знову тільки ті набори наборів є значними, які перетинають порогову підтримку, а такі - OP, OB, PB та PM.

**Крок 6.** Тепер скажімо, ми хотіли б шукати набір з трьох елементів, які купуються разом. Ми будемо використовувати пункти, знайдені на кроці 5, і створимо набір з 3 елементів.

Щоб створити набір з 3 елементів, потрібно інше правило, яке називається самоз'єднання. Він говорить, що з пар OP, OB, PB та PM ми шукаємо дві пари з однаковою першою буквою, і об’єднуємо їх:

* OP та OB, це дає OPB
* PB і PM, це дає PBM

Далі ми знайдемо частоту для цих двох наборів.

Таблиця 2.5.

Таблиця частот

|  |  |
| --- | --- |
| **Набір** | **Частота** |
| OPB | 4 |
| PBM | 3 |

Знову застосувавши правило порогу, ми виявили, що OPB є єдиним значним набором елементів.

Отже, набір з 3 предметів, який був придбаний найчастіше, - OPB.

Приклад, який ми розглянули, був досить простим, і видобуток частих предметів у товаристві зупинено на 3 пунктах, але на практиці існує кілька десятків предметів, і цей процес може тривати багато предметів. Припустимо, що ми отримали значні набори з трьома елементами як OPQ, OPR, OQR, OQS та PQR, і тепер ми хочемо створити набір з 4 елементів. Для цього ми розглянемо набори, які мають перші два алфавіти, тобто загалом

OPQ та OPR дають OPQR

OQR та OQS дають OQRS

Взагалі, ми повинні шукати набори, які тільки відрізняються в їхній останній букві / предметі.

Тепер, коли ми розглянули приклад функціональності алгоритму Апріорі, давайте сформулювати загальний процес.

Весь алгоритм можна розділити на два етапи:

**Крок 1:** застосуйте мінімальну підтримку, щоб знайти всі часто використовувані набори з елементами k у базі даних.

**Крок 2.** Використовуйте правило самопідключення, щоб знайти часті множини з елементами k + 1 за допомогою часто використовуваних k-елементів. Повторіть цей процес з k = 1 до моменту, коли нам не вдається застосувати правило самопідключення.

Такий підхід до розповсюдження частих елементів, що належить до одного, називається підходом "знизу вверх".

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), у разі використання HashMap.

* 1. **Видобування асоціативних правил**

До цих пір ми розглянули алгоритм Апріорі по відношенню до генеруванні частих наборів. Існує ще одне завдання, за допомогою якого ми можемо використовувати цей алгоритм, тобто ефективно знаходити асоціативні правила. [26-27]

Щоб знайти асофіативні правила, нам потрібно знайти всі правила, що мають підтримку, більшу за порогову підтримку та довіру, більшу за порогову довіру.

Але як ми знаходимо це? Одним із можливих шляхів – це перебір всіх можливих асоціативних правил та обчислення підтримки та впевненості для кожного правила. Потім просканувати правила, які не підтримують порогову підтримку та довіру. Але це обчислювально дуже складний і неефективний процес, оскільки кількість всіх можливих правил асоціації зростає експоненціально з кількістю предметів.

Враховуючи, що в наборі I є n пунктів, загальна кількість можливих правил асоціації становить 3 ^ n - 2 ^ (n +1) + 1.

Ми також можемо використовувати інший спосіб, який називається двоетапним підходом, для пошуку ефективних правил асоціації.

Двоступеневий підхід:

Крок 1: генерація частотних наборів елементів: знайдіть всі набори предметів, для яких підтримка перевищує порогову підтримку відповідно до процесу, який ми вже бачили раніше в цій статті.

Крок 2: генерація правил: створюйте правила з кожної частинки, використовуючи бінарний розділ часто використовуваних предметів і шукайте тих, що мають високу впевненість. Ці правила називаються правилами кандидатів.

Давайте розглянемо наш попередній приклад, щоб отримати ефективне правило об'єднання. Ми виявили, що OPB є частим комплектом. Тому для цієї проблеми, крок 1 вже зроблено. Отже, давайте розглянемо крок 2. Усі можливі правила, що використовують OPB, є:

OP\longrightarrowB, OB\longrightarrowP, PB\longrightarrowO, B\longrightarrow OP, P\longrightarrowOB, O\longrightarrowPB

Якщо X являє собою частий набір елементів з k-елементами, то існують правила асоціації кандидатів 2 ^ k-2.

Плюси алгоритму:

Легкий у реалізації та розумінні.

Може бути використаний для величезних даних.

Мінуси алгоритму:

Іноді, потрібно знайти величезну кільуість правил кандидатів, що є обчислювано затратно.

Підрахуном підтримки також затратний по ресурсах, адже потрібно пройтись по всій базі.

* 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 [13].

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.

Алгоритм AprioriTid 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:

 (2.1)

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 [13].

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.

* 1. **Опис можливих модифікацій**

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

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.

Цей підхід можна спробувати для вище перелічених методів, а також спробувати даний алгоритм при різних значеннях коефіцієнтів підтримки та достовірності.

Ще одним із можливих варіантів модифікації є розділ множини факторів I на дві підмножини:

1) фактори, які мали місце до виникнення асоціації Si;

2) фактори, що стали результатом виникнення асоціації Di.

Асоціативні правила, що містять лише підмножину факторів 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 факторів 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.

* 1. **Висновок до розділу 2**

Здійснено обгрунтування та порівняльний аналіз алгоритму, 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 зменшить витрати часу при знаходженні асоціативних правил. Також є можливим впровадження даного підходу що 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 це множина із факторами які мали місце до виникнення асоціацій і друга – множина із факторами що стали результатом виникнення асоціацій, на кожному кроці відсікається перша множина, адже вона не містить корисної інформаці.

# Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІя АЛГОРИТМУ apriori



## Загальна постановка задачі

Реалізувати програмний засіб, за допомогою якого можна здійснювати інтелектуальний аналіз даних, а саме пошук асоціативних правил. Даний програмний засіб повинен надавати можливість для проведення аналізу вхідних даних та виведення результатів аналізу (знайдені асоціативні правила, їх кількість та швидкодію). Після виконання програма повинна формувати звітність на основі результатів аналізу представлених даних. Результати будуть представлені безпосередньо на екрані із можливістю збереження їх у файл.

Провести дослідження ефективності алгоритму. На основі отриманих даних із експериментів визначити можливість видозміни алгоритму для збільшення ефективності його роботи. При позитивних результатах виконати модифікацію алгоритму.

## Специфікація вимог

**3.2.1. Вступ**

Призначення даного продукту – автоматичний аналіз великих наборів даних та пошук в них асоціативних правил, формування звіту.

Мета – реалізувати класичний алгоритм apriori та модифіковані його версії та порівняти результати роботи у вигляді програмного продуку.

**3.2.2. Загальний опис**

**3.2.2.1. Характеристики продукту**

Функції, що будуть реалізовані у програмі:

* Зчитування даних з файлу;
* Аналіз даних;
* Виведення результатів у вигляді таблиць та діаграм;
* Формування звіту про результати аналізу;

**3.2.2.2. Класи користувачів та їх характеристики**

В даній програмі передбачено лише один клас користувачів, тому функціонал не буде мати обмеження.

Програмний засіб призначений для бізнес аналітиків, щоб ефективно провести інтелектуальний аналіз даних та сформувати звіт замовнику.

**3.2.2.3. Середовище функціонування**

Апаратні вимоги:

* Частота процесора 1 GHz;
* Оперативна пам’ять 512 Mb;
* 10 Mb вільного простору на диску.

Системні вимоги:

* Операційна система Microsoft Windows XP або вище;
* .Net Framework версії 4.0 і вище;

**3.2.3. Характеристики системи**

**3.2.3.1. Пошук асоціативних правил для введених даних**

3.2.3.1.1. Опис і пріоритет

Аналіз введених даних, пошук залежностей. Виведення результатів тестування у зручному для користувача вигляді, у вигляді таблиць та діаграм, із можливістю збереження на локальний диск.

Пріоритет – високий.

3.2.3.1.2. Послідовності дія/відгук

Користувач вибирає текстовий документ із даними, аналіз яких потрібно провести. Далі завантажує список характеристик написнувши на кнопку «Завантажити I». Потім задає коефіцієнти підтримки, достовірності та ліфта, при потребі завантажує причини виникнення ДТП. Натискає на кнопку «Розпочати» і у низу вікна з’являється анімація завантаження.

3.2.3.1.3. Функціональні вимоги

REQ-1.1: Зчитування даних із .txt формата;

REQ-1.2: Проведення аналізу;

REQ-1.3: Пошук асоціативних правил;

REQ-1.4: Виведення повідомлення про закінчення проведення аналізу;

REQ-1.5: Виведення результатів;

**3.2.3.2. Формування звіту результату**

3.2.3.2.1. Опис і пріоритет

Створення звіту, який буде відображати інформацію про результат пошуку залежностей. Звіт можна зберегти на диск у форматі txt.

3.2.3.2.2. Послідовності дія/відгук

Для збереження звіту потрібно натиснути кнопку «Зберегти звіт», після чого появиться діалогове вікно для вибору місця збереження файлу.

3.2.3.2.3. Функціональні вимоги

REQ-2.1: Формування звіту;

REQ-2.2: Збереження на диску у форматі txt;

**3.2.3.3. Перегляд результатів виконання**

3.2.3.3.1. Опис і пріоритет

Перегляд результатів виконання програми у вигляді діаграм та таблиці

3.2.3.3.2. Послідовності дія/відгук

Для перегляду результатів потрібно натиснути кнопку «Результати», після чого появиться нове вікно із результатами виконання.

3.2.3.3.3. Функціональні вимоги

REQ-3.1: Формування звіту;

REQ-3.2: Збереження на диску у форматі txt;

**3.2.4. Вимоги зовнішніх інтерфейсів**

**3.2.4.1. Користувацькі інтерфейси**

Інтерфейс програми повинен бути простим для розуміння, забезпечувати швидкодію відповідно до вимог та дозволяти зручно виконувати необхідні операції.

**3.2.4.2. Програмні інтерфейси**

* Бібліотека Microsoft.Office.Interop.Excel;
* Бібліотека Xml.Serializer;

**3.2.5. Інші нефункційні вимоги**

**3.2.5.1. Вимоги продуктивності**

Швидкість проведення аналізу буде прямо залежати від кількості завантажених даних, коефіцієнтів роботи алгоритму та від потужності процесора.

**3.2.5.2. Вимоги безпеки**

Немає вимог безпеки.

**3.2.5.3. Атрибути якості програмного продукту**

Програма реалізована із мінімальним основним набором функції, саме тому вона зрозуміла та зручна для використання користувачам, які з нею ніколи не працювали;

Ніяким чином не впливає на роботу інших програм.

**3.2.5.4. Інші вимоги**

Необхідна наявність .Net Framework версії 4.0 і вище.

## Використані технології

Оскільки основним завданням проекту буде аналіз роботи алгоритму то було обрано технологію .Net, середовище розробки Microsoft Visual Studio, мова C# та WPF.

Для зберігання поточного стану розробленої системи використано систему контролю версій Git - це система керування версіями для відстеження змін у комп'ютерних файлах та координаціїіроботи над цими файлами серед декількох людей. Він використовується, перш за все, для управління вихідним кодом у розробці програмного забезпечення [8], але йогоіможнаівикористовувати для відстеження змініу будь-якомуінаборі файлів. В якості розподіленої системи перегляду-контролю вонаіспрямованаіна швидкість, цілісністьіданих та підтримку розподілених, нелінійних робочих процесів.

.NET Framework є основою програмного забезпечення, розроблена корпорацією≥Майкрософт, яка працює≥в основному на Microsoft Windows. Вона містить велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує мовну сумісність (кожна мова може використовувати код, написаний на інших мовах) на кілька≥мов програмування. Програми, написані для .NET Framework виконуються в програмному≥середовищі (на відміну від апаратної середовища) під≥назвою Common≥Language Runtime (CLR), віртуальна≥машина додаток, яке надає≥такі послуги, як безпека, управління≥пам'яттю і обробки винятків.

C # — є об’єктно-орієнтованою мовою≥програмування≥яка охоплює строгу типізацію, імперативність, декларативність, функціональні, загальні, об'єктно-орієнтовані (на основі класів), компонент-орієнтоване≥програмування дисциплін. C # є одним з мов програмування, призначених для Common Language Infrastructure.

Для малювання діаграм використано бібліотеку DataVisualization, яка дозволяє легко та прозоро малювати різного роду діаграми.

Графічний інтерфейс буде≥реалізовано на основі технології WPF (Windows Presentation Foundation). Дана≥технологія призначена для організації≥взаємодії WIN-програм≥з користувачем, тобто WPF≥забезпечує≥інтерфейс≥користувача.

## Реалізація інтерфейсу користувача програмного засобу для роботи алгоритму apriori

Розроблений інтерфейс користувача складається із мінімального набору компонентів, що робить його дуже зручним та простим у користуванні, в той самий час, даний інтерфейстдаєтзмогу користувачу в повній мірітзастосувати весь функціоналтрозробленого засобу.

Інтерфейс користувача складається з основного вікна та додаткового (див. Рис. 3.1. та Рис. 3.2.). Основне вікно застосунку дозволяє користувачу завантажити список транзакцій(список із ДТП), завантажити характеристики ДТП, завантажити причини виникнення ДТП, а також встановити коефіцієнти підтримки достовірності та ліфта. Також, на головному вікні користувач має можливість почати процес аналізу великих даних алгоритмом apriori, та побачити результати виконання алгоритму результати (кількість знайдених правил чис виконання, кількість транзакій тощо.).

Вікно «Результати» відповідає за відображення результатів програми та порівняння їх із попередніми значеннями. Дане вікно складається дане вікно складається із трьох діаграм, таблиці та кнопки «Зберегти»(Рис. 3.2.). Таблиця представляє список із результатами виконання програми та такими колонками як: «Спроба №» - показує номер спроби, «Час виконання» - час виконання анлізу для спроби, «Кількість транзакцій» - кількість транзакцій у початковому списку, «Кількість правил» - кількість знайдених асоціативних правил.

Таблиця «Кількість знайдених правил» показує відношення у відсотках скільки правил було знайдено для кожної спроби.

Таблиця «Кількість транзакцій» показує скільки транзакцій було в початковій базі даних.

Таблиця «Час виконання» показує час виконання для кожної спроби.

Якщо користувачцвводить неправильне значення змінної (наприклад, для змінної типу int – стрічку), програмацнецдасть змоги зберегтиезміни і попередить проецеекористувача(Рис. 3.3.).

Отже розглянувши всі ці результати можна легко зробити висновки про результати роботи алгоритму як для мофікованих режимів так і для стандартного вигляду.

## Розробка архітектури системи для роботи алгоритму apriori

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

* FileReader відповідає за зчитування даних із файлів та конвертання їх у масив. Після відкриття файлу з’являється діалогове вікно для вибору .txt файлу.
* 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 «transactions», масив «itemsD» мінімальне значення вірогідності, підтримки та ліфта. Та після обробки повертає результат типу Output.
* AprioriInThread – клас який забезпечує виконання алгоритму в окремому потоці щоб не блокувати користувацький інтерфейс, містить один метод «ProcessInThread» який повертає результат Task<Output>.
* Клас Result містить інформацію яка необхідна для побудови графіків та списку у вікні з результатами.

Також було розроблено UML діаграму прецедентів(Рис. 3.5) на якій зображено відношення акторів та прецедентів в системі.

## Реалізація алгоритму apriori для аналізу ДТП

Алгоритм apriori було реалізовано згідно наведеної блок-схеми (Рис. 3.4).

Список транзакцій завантажується із .txt файла та обробляється до потрібного вигляду, це двовимірний масив стрічок, за допомогою класу FileReader. Далі за таким самими принципом завантажується список характеристик та наслідків.

Список транзакцій разом із коефіцієнтами подається на вхід функції ProcessTransactions. Всередині цієї функції відбувається покрокова генерація кандидатів та частих наборів.

Під час генерації кандидатів обраховується їхня кількість відносно всіх транзакцій, та якщо використовується модифікована версія із завантаженими причинами то відсікаються кандидати у яких немає цих причин, завдяки такому підходу на кожній ітерації вдається відсікнути багато зайвих кандидатів і тому виконання алгоритму стає набагато швидшим.

Під час генерації частих наборів обраховується «підтримка» та порівнюється із мінімально допустимою, якщо ця умова виконується кандидат додається до частих наборів.

Усе це повторяється поки множина кандидатів не буде порожньою.

Після цього відбувається генерація асоціативних правил на основі частих наборів. Далі відбувається генерація «міцних» правил в методі GetStrongRules, де відбувається обрахунок вірогідності, та якщо використовується модифікована версія обраховується значення ліфта, такий підхід дозволяє відсікати абсурдні або випадкові правила та дозволяє збільшити точність виконання алгоритму.

Далі метод ProcessTransactions повертає список Output об’єктів з яких в подальшому можна згенерувати звіт та порівняльні діаграми.

## Висновки до розділу 3

Ключовим завданням постає розробка програмного засобу для проведення аналізу даних алгоритмом apriori. По суті програма буде працювати у певній логічній послідовності, отже варіантів дій користувача не так і багато, що робить користування просте та зрозуміле. Користувачу представляється лише вибір даних, який слід аналізувати і результати для подальшого використання.

Так як ціль проекту включає в собі інтелектуальний аналіз даних то технологія Microsoft .NET чудово підходить для написання прикладної програми. Для побудови інтерфейсу використовуватиметься технологія WPF, яка дозволить без проблем створити цікавий вигляд програми.

У цьому розділі детально описано специфікацію вимог, реалізовано інтерфейс, архітектуру системи та реалізовано алгоритм.

# Розділ 4. ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ОТРИМАНИХ ДАНИХ

## 4.1. Опис процесу дослідження

Поданий алгоритм apriori використовується для аналізу дорожньо-транспортних пригод, а саме пошуку асоціативних зв’зків між наборами даних які стали причинами ДТП.

При застосуванні пошуку асоціативних правил для аналізу ДТП виникають проблеми:

1. Збільшення потужності множини всіх можливих ДТП.
2. Збільшення часу для аналізу даних.
3. Низька інформаційність отриманого результату.

Запропоновано наступні модифікації для розв'язання цих проблем:

1. Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини:[3]

* *Is* – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами (стан алкогольного сп’яніння, технічний стан автомобіля, вік водія, вік авто тощо).
* *Id* – множина характеристик, дані якої для нашого випадку будуть цікавими (місце проживання – село/місто, освіта, колір машини, кількість пасажирів, порядок власника авто тощо).

Асоціативні правила, що містять лише підмножину із множини Is не містять корисної інформації, тому на кожному кроці після генерації кандидатів можна відсікати кандидатів, для яких Ck ∩ Id = null.

У результаті застосування такого підходу збільшується швидкість роботи алгоритму та ступінь інформаційності отриманих асоціативних правил.

1. Додати ще одну характеристику асоціативних правил Ліфт (англ. lift) - це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку в цілому.[4] Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил, отже швидкість алгоритму повинна зрости адже не потрібно здійснювати наступні кроки по вже відкинутих правилах.

Отже, для якісної оцінки запропонованих модифікацій потрібно

згенерувати базу даних із причинами які спровокували виникнення дорожньо-транспортних пригод.

Перш за все, потрібно здійснити аналіз стандартного алгоритму без модифікацій.

База даних складається із таких характеристик:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Кількість власників (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);
* Стать (чоловіча, жіноча);

Другим етапом дослідження є порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Третім етапом буде порівняння роботи стандартного алгоритму із запропонованою другою модифікацією.

На наступному етапі можна порівняти між собою три попередні варіанти.

На завершення потрібно підібрати найкращі варіанти коефіцієнтів алгоритму, які б показували найкращі результати в швидкодії та точності пошуку асоціативних правил.

## 4.2. Аналіз ароботи алгоритму без модифікацій

Після генерації даних було здійснено аналіз роботи алгоритму без модифікацій.

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori без додаткових модифікацій для дорожньо-транспортних пригод наведені в Таблиці 4.1.

*Таблиця 4.1.*

Результати виконання розробленого засобу

|  |  |
| --- | --- |
| Номер досліду | Час виконання, мс |
| 1 | 40 409 |
| 2 | 40 407 |
| 3 | 42 354 |
| 4 | 45 373 |
| 5 | 41 295 |
| 6 | 45 573 |
| 7 | 43 401 |
| 8 | 42 563 |
| 9 | 45 765 |
| 10 | 43 642 |

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 42 878.2мс.

Також, аплікація згенерувала графік на якому видно, що дані підтверджено. Результат представлений на рис. 4.1.

Таким чином, дані які представлено вище можна вважати еталоном для нашого дослідження, так як це результати роботи алгоритму без будь яких додаткових втручань.

Отже, середнє значенням часу виконання алгоритму дорівнює 42 878.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила таких як:

{чоловіча –> місто}, вірогідність: 0.692307692307692

{місто –> чоловіча}, вірогідність: 0.580645161290323

{місто –> alco 1.0-1.5}, вірогідність: 0.580645161290323

{alco 1.0-1.5 –> чоловіча}, вірогідність: 0.75

Також знайдено 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

## 4.3. Аналіз роботи алгоритму із модифікаціями

Для початку дослідимо роботу алгоритму для модифікації №1 – Розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини.

Для множини Id – множина із характеристиками що описують ДТП, було обрано такі категорії:

* Колір (білий, чорний, червоний, зелений, жовтий);
* Кількість пасажирів в салоні на момент ДТП (пасажири 0, пасажири, пасажири 2, пасажири 3, пасажири 4, пасажири 4+);
* Кількість власників (власник 1, власник 2, власник 3, власник 4+);
* Вік автомобіля (1-5, 6-10, 11-15, 16-20);
* Освіта (вища, середня, базова);
* Місце проживання (місто, село);
* Стать (чоловіча, жіноча);

Для множини Is – множина характеристик, дані якої для нашого випадку є абсурдними або загальновідомими фактами

* Пора року (літо, осінь, зима, весна);
* Пора доби (ранок, обід, вечір, ніч);
* Рівень алкоголю в крові (alco 0, alco 0.2-0.5, alco 0.6-0.9, alco 1.0-1.5, alco 1.6-2.8);
* Вік водія (15-17, 18-21, 22-25, 26-30, 31-35, 36-40, 41-45, 46-50);

Результати застосування розробленого засобу для аналізу роботи алгоритму apriori із модифікацією №1 для дорожньо-транспортних пригод наведені в Таблиці 4.2.

*Таблиця 4.2.*

Результати виконання алгоритму із модифікацією №1

|  |  |
| --- | --- |
| Номер досліду | Час виконання, мс |
| 1 | 37 409 |
| 2 | 38 407 |
| 3 | 39 354 |
| 4 | 38 373 |
| 5 | 37 295 |
| 6 | 38 573 |
| 7 | 36 401 |
| 8 | 38 563 |
| 9 | 39 765 |
| 10 | 37 642 |

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 38 178.2мс.

Також, результати можна переглянути на графіку. Результат представлений на рис. 4.3.

Середнє значенням часу виконання модифікованого алгоритму №1 дорівнює 38 178.2мс, для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 та мінімальне значення вірогідності 0.5.

Отже, можна побачити, що час роботи алгоритму скоротився на 11% при цьому кількість знайдених частих наборів і «міцних» правил не зменшилась.

Приступимо до аналізу роботи алгоритму для модифікації №2 – додавання ще однієї характеристику асоціативних правил Ліфта.Обраховуючи Ліфт на кожному кроці алгоритму можна здійснити більше відсікання абсурдних або випадкових груп правил. Проте перш ніж здійснити аналіз потрібно уточнити що застосувати Ліфт на етапі генерації кандидатів не вийшло так як логіка алгоритму не дозволяє цього зробити, ліфт можна застосувати при генерації асоціативних правил.

Рзультати застосування розробленого засобу для аналізу роботи алгоритму apriori із модифікацією №2 для дорожньо-транспортних пригод наведені в Таблиці 4.3.

*Таблиця 4.3.*

Результати виконання розробленого засобу

|  |  |
| --- | --- |
| Номер досліду | Час виконання, мс |
| 1 | 41 309 |
| 2 | 42 107 |
| 3 | 42 454 |
| 4 | 44 373 |
| 5 | 42 215 |
| 6 | 45 573 |
| 7 | 42 421 |
| 8 | 42 563 |
| 9 | 44 565 |
| 10 | 43 742 |

Таким чином, середній час виконання алгоритму, виміряний розробленим засобом становить 43 132.2мс.

Також, аплікація згенерувала графік на якому видно, що дані підтверджено. Результат представлений на рис. 4.4.

Як видно із результатів виконання алгоритму дані не сильно відрізняються від еталонних(результати які було отримано із базового алгоритму) в деяких випадках результати навіть гірші. На практиці додавання характеристики Ліфта до роботи алгоритму нічого не ало навіть погіршило результати. Проте це тільки для нашого конкретного випадку і бази даних із дорожнього-транспортними пригодами можливо приінших випадках результатит були б інакшими.-

Отже, середнє значенням часу виконання алгоритму дорівнює 43 132.2 мс

для даних обсягом 230мб та кількістю транзакцій 1 000 000 одиниць, було знайдено 4 міцних правила та 11 частих наборів, які перевищують значення мінімального значення підтримки. Мінімальне значення підтримки дорівнює 0.4 мінімальне значення вірогідності 0.5 та мінімальне значення Ліфта дорівнює 0.9.

## 4.4. Висновки до розділу 4

У цьому розділі було проведено дослідження алгоритму Apriori та його модифікацій. Дослідження здійснено в такій послідовності: перш за все, було проведено аналіз базового алгоритму без модифікацій, далі було проведено порівняння роботи стандартного алгоритму із запропонованою першою модифікацією – розбиття множини всіх можливих транспортних пригод *I = {Is, Id*} на дві підмножини, третім етапом було порівняння роботи стандартного алгоритму із запропонованою другою модифікацією. Останнім етапом було порівняння між собою трьох попередніх варіантів.

Отже, після здійснення досліджень можна сказати що запропонований варіанти модифікації №1 є найкращим у нашому випадку, за цим методом далось скоротити час виконання алгоритму на 11%, при цьому не втративши жодної важливої інформації.

Варіант модифікації №2 не вдалось реалізувати як було описано в теорії, тому що це руйнує логіку алгоритму, проте можна застосувати при генерації асоціативних зв’язків але результати для нашого випадку, аналіз ДТП, були гіршими за еталон.

# Висновки

Магістерська кваліфікаційна робота присвячена дослідженню та розробці алгоритму Apriori, та введенню можливих модифікацій для кращої роботи алгоритму.

Було здійснено порівняльний аналіз роботи класичного алгоритму Apriori та його модифікацій. Опрацьовано дані вказаним алгоритмом та оцінено швидкість роботи алгоритму, завантаженість на ресурси, кількість знайдених асоціативних правил для база даних із дорожньо-транспортними пригодами. Вироблено на основі обчислювальних експериментів пропозицію для модифікації алгоритму.

Для програмної реалізації створеного алгоритму використано технології .Net, мову C#, технологію WPF для створення інтерфейсу користувача та інтеграційне середовище розробки Visual Studio 2015.

Було проведено дослідження розробленого засобу. Дослідження складається з таких частин: дослідження класичного алгоритму без модифікацій, дослідження модифікованого алгоритму за першим способом, дослідження модифікованого алгоритму за другим способом, дослідження роботи алгоритму при різними значеннями коефіцієнтів та порівняльний аналіз вище перелічених досліджень.

Отже, після здійснення досліджень можна сказати що запропонований варіанти модифікації №1 є найкращим у нашому випадку, за цим методом далось скоротити час виконання алгоритму на 11%, при цьому не втративши жодної важливої інформації.

Варіант модифікації №2 не вдалось реалізувати як було описано в теорії, тому що це руйнує логіку алгоритму, проте можна застосувати при генерації асоціативних зв’язків але результати для нашого випадку, аналіз ДТП, були гіршими за еталон

Науковою новизною даного дослідження є розроблена модифікація для базового алгориту, яка дозволяє проводити аналіз безиких даних швидше не втративши при цьому точності.

Практична цінність отриманих результатів полягає у тому, що реалізоване програмне забезпечення дозволяє проводити аналіз великих даних, а для нашого випадку для попередження виникнення нових ДТП.

За результатами дослідження опубліковано матеріали доповіді на Міжнародній науковій інтернет-конференції "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення"(випуск 33) з доповіддю на тему «Модифікація алгоритму Apriori для аналізу дорожньо-транспортних пригод».

Проведено економічну характеристику розробленого програмного продукту. Згідно проведених розрахунків можна стверджувати, що розроблювані засоби для аналізу роботи алгоритму Apriori є кращою від її аналогу за економічними показниками. Термін окупності витрат на проектування становить всього 1 рік та 5 місяців.