Market-basket Analysis of MeDAL Dataset

Name	: Ahmad Luqman Al Hakim
Surname	: bin Shamsudin
Degree	: Master of Science in Data Science and Economics
Student ID	: 966034
Instructor	: Professor Dario Malchiodi
GitHub link	: https://github.com/luqmanshamsudin/market-basket-medal

Declaration

Introduction

Market-basket analysis, a widely utilised technique in data mining, plays a crucial role in uncovering associations and patterns within large-scale relational datasets. Market-basket analysis can offer valuable insights into healthcare by identifying frequently co-occurring medical terms within text data. Analysis of medical texts has the potential to offer a better understanding of disease patterns, treatment options, and potential adverse drug reactions.

In this paper, I focus on applying market-basket analysis to the MeDAL dataset, a comprehensive collection of medical texts. By analysing the text column of the dataset as baskets and considering individual words as items, I aim to discover frequent item sets and meaningful associations amongst medical terms. For instance, discovering co-occurring terms such as "hypertension" and "diuretics" may suggest a relationship between these concepts in clinical practice.

Moreover, identifying frequent associations can support knowledge discovery and hypothesis generation in medical research. By employing market-basket analysis on the

MeDAL dataset, this paper aims to demonstrate the usefulness of this approach by leveraging large-scale text data to reveal meaningful relationships among medical concepts, ultimately leading to improved healthcare outcomes.

Section 2: Dataset

The MeDAL (Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding) dataset comprises 14,393,619 articles (~15 GB) and three abbreviations per article on average. However, the primary implementation of the algorithm only uses 0.001 of the whole dataset, amounting to 28,809 rows of text data. The dataset was created by (Wen et al., 2020) using abstracts published on PubMed, a search engine that indexes scientific publications in the biomedical domain. The dataset contains three columns which are text, location and label.

The column text contains the abstracts, the column location contains the index of the ambiguated word, and the column label contains the word that was ambiguated. Shown below are a row from the dataset as an example:

Text	Location	Label
'alphabisabolol has a primary antipeptic action depending on dosage which is not caused by an alteration of the phvalue the proteolytic activity of pepsin is reduced by percent through addition of bisabolol in the ratio of the antipeptic action of bisabolol only occurs in case of direct contact in case of a previous contact with the **ATP** the inhibiting effect is lost'	'56'	'substrate'

Section 3: Methodology

The algorithm considered to be implemented to find frequent itemsets in text medical abstracts is the Apriori algorithm (Agrawal & Srikant, 1994). It operates on the principle of the "Apriori property," which states that if an itemset is frequent, then all of its subsets must also be frequent.

To implement this algorithm, the medical abstracts need to be pre-processed. Data pre-processing aims to improve the quality and suitability of the text data for subsequent analyses or modelling tasks. The primary pre-processing function, pre-process (), is defined in Python. It takes a row of text as input and performs several operations to clean and transform the text. First, the function converts the text to lowercase, ensuring consistency in text representation. Next, punctuation marks and the string are removed using the translate()

method.punctuation parameter. To eliminate commonly occurring and less informative words, the function removes stopwords using the NLTK's English stopword list. The text is then tokenised into individual words using the word_tokenize() method. The resulting word tokens are filtered by excluding stopwords, creating a list of meaningful words. The function employs lemmatisation for further normalisation, a process that reduces words to their base or dictionary form. The WordNet lemmatiser from NLTK is used to perform this operation on the filtered words. Finally, the pre-processed text is returned as a list of unique lemmatised words, ensuring that duplicate words are removed.

Then, the Apriori algorithm is implemented to find frequent itemsets within the baskets of words, which is an abstract. The implementation steps are performed using scalable techniques from Spark. The function takes an RDD (Resilient Distributed Dataset) as input, representing the dataset to be analysed. Functions from Spark such as ”map”, ”flatMap”, ”reduceByKey” are also used. The implementation also considers the support threshold based on the perc_threshold parameter. This threshold is calculated by multiplying the total count of records in the RDD by the specified percentage. The RDD is then pre-processed using the pre-process function. Next, I create two vocabularies: integer2word and word2integer. These dictionaries facilitate the indexing and mapping of words to integer values, enabling efficient itemset representation. The RDD is indexed using the create_indexing() function, which replaces each word in the RDD with its corresponding index value based on the word2integer dictionary.

The algorithm begins by scanning the dataset to identify frequent individual items, forming the initial set of frequent itemsets (i.e., frequent singletons). It then iteratively generates larger itemsets by joining pairs of frequent itemsets from the previous iteration. These larger itemsets are checked against the dataset to determine their frequency. The algorithm continues this process until no new frequent itemsets can be generated. The result is a collection of frequent itemsets, which can be further analysed or used to generate association rules that reveal significant relationships between items in the dataset. I implemented the Apriori algorithm in this project to identify itemsets up to frequent tripletons.

In the final part of the implementation, I map the integer-coded itemsets to their corresponding words using the integer2word dictionary and return the results in a readable format. Each result consists of an itemset and its support count.

Finally, I ran an experiment to improve the sensibility of the generated doubletons and tripletons. Theoretically, increasing the input RDD size and minimum support would generate better results. However, I do not have the computational resources to test this. Nevertheless, I attempted to see how the frequent itemsets changed when I tweaked one of the parameters separately. The changes are then evaluated with regard to their sensibility from the medical perspective. The results of the initial frequent itemsets generated are presented in the next section. I also discuss the outcomes of the experiment in the next section.

Section 4: Results

The first table below shows 5 randomly draw samples of the frequent itemsets identified with 0.001 of the whole dataset and 10% minimum support threshold.

Frequent Singletons	Frequent Doubletons	Frequent Tripletons
[('potential', 1323), ('could', 1294), ('associated', 1704), ('difference', 1292), ('type', 1350),	(('also', 'study'), 1107), (('patient', 'study'), 1222), (('cell', 'study'), 1113), (('study', 'result'), 1359), (('effect', 'study'), 1159)]	N/A

Due to my limited computational resources, it is impossible to test how the generated frequent itemsets improve by increasing the input data size and the minimum support threshold. Therefore, I attempt to show the difference by reducing both parameters instead.

The second table shows the frequent itemsets generated after I decreased the size of the RDD to 0.0001 of the original dataset, and the minimum support threshold was reduced to 5%. As previously, there are no frequent tripletons generated.

Frequent Singletons	Frequent Doubletons	Frequent Tripletons
('cancer', 71), ('presence', 98), ('increased', 189), ('approach', 83), ('similar', 107)	(('clinical', 'study'), 70), (('cell', 'protein'), 82), (('result', 'may'), 86), (('one', 'result'), 72), (('result', 'however'), 93), (('cell', 'result'), 92),	N/A

Section 5: Discussion

As shown in the results section, the implementation of Apriori algorithm is useful to generate frequent itemsets from textual data such as medical abstracts from MeDAL dataset.

There are several main appeals to using the Apriori algorithm to identify frequent itemsets in medical texts such as the MeDAL dataset. Firstly, the Apriori algorithm provides transparent and interpretable results. It generates frequent itemsets, sets of items that frequently co-occur in the dataset. These itemsets can provide valuable insights into associations and patterns within medical texts, allowing for a better understanding of relationships between medical terms or concepts. Secondly, the Apriori algorithm is scalable and can handle large datasets efficiently. It utilises an iterative approach that gradually explores the itemset space, avoiding the need to generate all possible itemsets upfront. This makes it suitable for analysing extensive medical text datasets, such as the MeDAL dataset, without significant performance issues.

However, the Apriori algorithm can be computationally expensive, especially when dealing with large datasets or datasets with high-dimensional itemsets. It requires multiple passes over the data and involves frequent itemset generation and candidate pruning. This computational overhead can pose challenges when working with extensive medical text datasets, potentially affecting the algorithm's runtime and scalability.

The Apriori algorithm considers itemsets as sets of items without considering their sequential or temporal order. In medical texts, the order of terms or events can often carry important information, such as the progression of a disease or treatment timeline. By disregarding sequence information, the Apriori algorithm may overlook meaningful associations or dependencies based on the order of items in medical texts.

To conclude, the Apriori algorithm can be useful for recognising patterns in medical texts such as used in this project. However, it can produce sensible results only if expanse computational resources are available, as conjectured by the experiment results. Otherwise, the results produced are not very sensible. This is a crucial aspect to consider by programmers trying to implement this and attempting to make contributions to the health sector, as anything in this sector deals with human life.

References

Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499.

Wen, Z., Lu, X. H., & Reddy, S. (2020). MeDAL: Medical Abbreviation Disambiguation Dataset for Natural Language Understanding Pretraining. *Proceedings of the 3rd Clinical Natural Language Processing Workshop*, 130–135. https://doi.org/10.18653/v1/2020.clinicalnlp-1.15