الجمهورية الجزائرية الديمقراطية الشعبية

PEOPLE’S DEMOCRATIC REPUBLIC OF ALGERIA

وزارة التعليم العالي و البحث العلمي

Ministry of Higher Education and Scientific Research

جامعة غرداية

University of Ghardaia

كلية العلوم و التكنولوجيا

Faculty of Science and Technology

قسم الرياضيات و الإعلام الآلي

Department of Mathematics and Computer Science

THESIS

Presented for the degree of **Master**

In: Computer Science **Specialty**: Intelligence System for Knowledge Extraction

By: Azeddine Hadj Kouider

Theme

Cache Management based on Frequent Patterns

Jury members

M.	Slimane Oulad Naoui	MCB	Univ. Ghardaia	President
M.	Youcef Mahjoub	MAA	Univ. Ghardaia	Examiner
M.	Kerrache Chaker Abdelaziz	MCB	Univ. Ghardaia	Supervisor

College year : 2019/2020

بسم الله الرحمن الرحيم
١٤٢٠ م

Dedicated to

My Dear Mother

The most precious person to me,who always supported me and took are of me when i was down and nearly giving up, thanks mom for being by my side and hope you will live long and keep blessing me with your kindness...

My Father

The person who always got my back and pushed me to go further he always pushed me to not giving up and made sure that i never needed anything my whole life thanks a lot dad for being by my side and i pray Allah to keep you safe and healthy as long as you live.

My Family

my grandparents who prayed for me and guided me with their wisdom and blessing i wish u along life so u could see the work you putted in me flourish and to my uncles and auntie who checked on me all the time and made sure am doing okay and special auntie aicha who cared about me a lot and offered me a lot of support may god bless with what ever you desires and last but not least my cousins and specially abdelhak who got my back in what ever i did and supported my decisions .

My Sisters and Brother

Lina, Nour and Fares the greatest gift my parents gave me and a true blessing from god thanks for keeping me laughing and loving me all this years.

My dear brother from another mother

Abderrahim Sehil, the one and only, who supported me and made me go crazy, the one who followed me through all the insane things i did through the years we spent together, and thanks for his parent who took care of me as their own son and maybe more. thanks uncle Azeddine for always cheering me up and keeping me close to Allah and for the great lesson to always be calm and think before you say anything, and to his mother who always prayed for me and made good food and sweets whenever i am around.

My Special Friends

Charaf , Derbali , Safi-eddine , Mohamed and Souhir, thanks for all the time we spent together and all the fun we had you guys took away the stress and pain of this journey, much love and much appreciation !!

and

I thank you all, cause without you this work wouldn't be possible.

Cordially

Azeddine.

Acknowledgment

“Words fly away, writings remain”

In the name of "ALLAH", The most beneficent and merciful who gave as strength and knowledge to complete this thesis.

Firstly, I would like to express my sincere sense of gratitude to my supervisor Mister **Kerrache Chaker Abdleaziz** who offered his continuous advice and encouragement throughout the course of this thesis. I thank him for the guidance and great effort he puts into training me in the scientific field.

I am deeply grateful to all members of the jury for agreeing to read this manuscript and to participate in the defense of this thesis.

I thank all the teachers who taught me in the five past years for the vast amounts of information.

For all those who participated in the development of this work.

Abstract

Cache management is a classic field of study where scientists and scholars tend to create or improve new caching schemes. Cache management algorithms tend to anticipate the requests and make it possible to bring the requested resources closer to the requesting sites and thus reduce latency times. Data Mining (DM)consists of many techniques extract significant knowledge from large datasets. Among data mining techniques association rules mining algorithms attempts to find interesting associations and relationships hidden in large amounts of data. Apriori algorithm is one of these methods which is used mostly for search optimization. Our work studies cache management using frequent patterns where we try to improve the LFU(least frequently used) replacement algorithm using frequent itemsets extracted using apriori algorithm. The results shows improvement in the hit and miss rate ratio by 4 % to 10%.

Keywords:

Data Mining (DM), Cache management, Frequent patterns, Apriori , LFU.

Résumé

La gestion du cache est un domaine d'étude classique où les scientifiques et les chercheurs ont tendance à créer ou à améliorer de nouveaux schémas de mise en cache,Les algorithmes de gestion du cache ont tendance à anticiper les requêtes et permettre de rapprocher les ressources demandées des sites demandeurs et réduire ainsi les temps des latences. La fouille de données (FD)se compose de nombreuses techniques pour extraire des connaissance utiles à partir de grands ensembles de données. Parmi les techniques d'exploration de données, les algorithmes d'extraction de règles d'association tentent de trouver des associations et des relations intéressantes cachées dans de grandes quantités de données, l'algorithme apriori est l'un de ceux qui a été principalement utilisé pour l'optimisation de la recherche. Notre travail étudie la gestion du cache en utilisant des modèles fréquents où nous essayons d'améliorer l'algorithme de remplacement LFU en utilisant des règles d'association extraites par l'algorithme apriori, nous avons amélioré le taux de réussite et d'échec de 4 % à 10 % .

Mots-clés:

Fouille de données, Gestion du cache, Motifs frquents, Apriori, LFU.

ملخص

تعد إدارة ذاكرة التخزين المؤقت أحد المجالات الكلاسيكية للدراسة حيث يميل العلماء و الباحثون إلى إنشاء أو تحسين مخططات التخزين المؤقت الجديدة، تميل خوارزميات إدارة ذاكرة التخزين المؤقت إلى توقع الطلبات وجعل من الممكن تقريب الموارد المطلوبة من المواقع الطالبة وبالتالي تقليل أوقات الاستجابة. التنقيب عن البيانات (DM) يتكون من العديد من التقنيات إستخراج المعارف المهمة من مجموعات البيانات ذات الأكوام الكبيرة. من بين تقنيات التنقيب عن البيانات ، تحاول خوارزميات التنقيب العثور على ارتباطات وعلاقات مثيرة للاهتمام مخفية في كميات كبيرة من البيانات ، وخوارزمية $apriori$ هي واحدة من تلك التي تم استخدامها في الغالب لتحسين البحث. يدرس عملنا إدارة ذاكرة التخزين المؤقت باستخدام أنماط متكررة حيث نحاول تحسين خوارزمية استبدال LFU باستخدام قواعد الارتباط المستخرجة بواسطة خوارزمية $apriori$، النتائج تبين تحسين نسبة الإصابة والفشل بنسبة ٤ إلى ١٠ .

الكلمات المفتاحية :

التنقيب في البيانات، ادارة ذاكرة التخزين المؤقت، ألانماط المتكررة، $(Apriori)$ ، (LFU)

Contents

List of Figures

List of Tables

Listings

Introduction

Today's world is characterized by the availability of enormous amounts of information and data, by processing this data using data mining, scientists discovered a lot of more useful information that could help improve marketing fraud detection and many others.

Data Mining (DM) is a science that, by exploring and analyzing huge amounts of information, aims to extract meaningful trends and patterns.

In this document, we are interested in Data Mining and more specifically frequent patterns that could be used to give or improve results.

Among the problems encountered when handling large amounts of data is the searching and delivering data, therefore the use of cache management can help reduce the size of these problems. Cache management, makes sure to get the data closer to the user and reduce data serving time and redundancy of the data.

Cache management came into existence for decreasing the gap between requesting the information and getting it.
in this thesis, we are exploring the study of cache management using frequent patterns, focusing mainly on using the apriori algorithm to extract frequent patterns then applying these frequent patterns on improving a certain scheme of cache replacement, after that I will try to test the improved version with different parameters (cache size, amount data passing through the cache) to understand the impact of each of them on the final result.

I have structured my manuscript in three main chapters:

- **The first chapter** begins with a definition of cache and cache management and its types. Then, it introduces the basics cache management schemes.
- **The second chapter** is devoted to the description of artificial intelligence, its different types, machine learning data mining, as well as the different properties that characterize each one of them, we also talked about frequent patterns, the apriori algorithm and its characteristics.
- **The third and last chapter** shows the experimental part of our work, it contains the various tools and software used, the dataset, the different parts of the code we used in this implementation and then the results.

Chapter 1

Cache Management

1.1 Introduction

This chapter is dedicated to cache, where we are going to introduce the notion of cache and cache management then some basic cache management schemes, we decided to focus on the cache placement schemes and then we list some algorithms with examples.

1.2 Cache Definition

Cache memories are intermediaries storage spaces associated with management architectures and algorithms, so the data when requested would be fast delivered, it saves a copy of the original data that remain in the main memory(database, hard disk, etc..) where its stored permanently.
When data is requested and found in the cache it is called a cache hit, otherwise, it is called cache miss. When there is a cache miss, the content requested is added to the cache so the next time when another request for a similar content will be satisfied.
Adding a cache node to a network as if the selected node has become a free storing place by adding it directly to the network.
”Caching is a mechanism for providing temporary storage to reduce bandwidth, server load, and response time”[8].
To select the node where the content is saved and used when needed with efficiency. We need schemes called cache placement schemes which are made to place content in a way that makes getting the content fast and not over-burden the network, in case the content is larger than the free space in the node some other scheme come in hand these are called cache replacement schemes, this schemes are responsible for finding the best content to remove and replace it with the newly requested content without effecting the network efficiency.

1.3 Cache Placement

Cache placement policy assigns where a particular copy of a memory bloc or item should be stored along the network path. cache placement algorithm are devised to coordinated and non-coordinated schemes

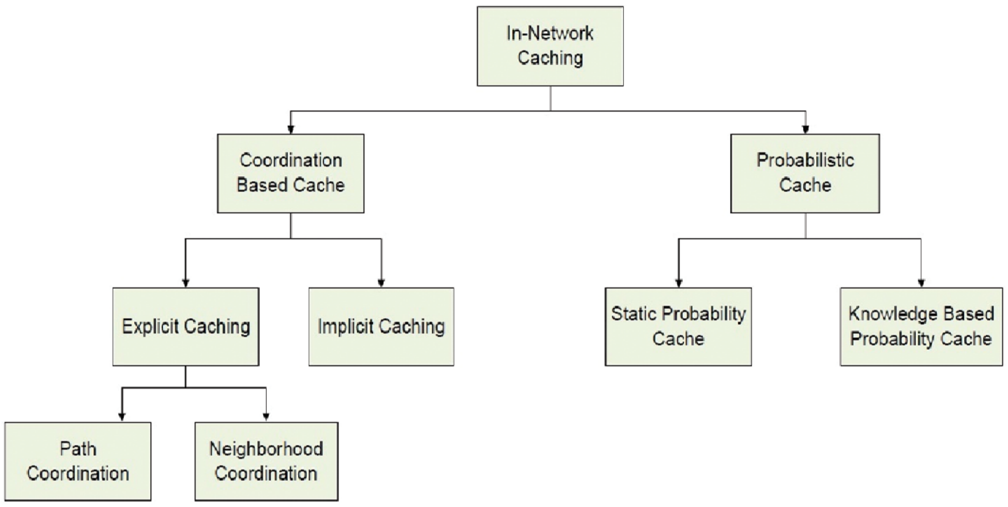


Figure 1.1: In-network cache taxonomy.[1]

1.3.1 Coordinated Schemes

Coordinated schemes are based on node coordination to minimize redundancy and improve cache diversity as [1] mentioned, these schemes use mobile cluster head or a road side unit to coordinate and incur control over the network, based on that we can divide these schemes into implicit and explicit as suggested by [1]. Coordinated caching helps for two reasons. First, coordination allows a busy cache to utilize a nearby idle cache Second, coordination balances the improved hit time achieved by increasing the replication of popular objects against the improved hit rate achieved by reducing replication and storing more unique objects. also it is efficient in reducing cache redundancy and improve the diversity of the cache and reduces cache traffic though it introduces high computational cost as concluded

1.3.1.1 Explicit Schemes

In this scheme it is well said that mobile cluster head or a road side unit are equipped with the prerequisite information of the network, cluster heads are defined by [9] as a selected node from a cluster of nodes, this nodes contain information about the cluster for example cache's state users access frequency and network topology, it also exchange a lot of information's in the same line.

Coordinated En Route Web Caching (CERWC) The coordinated en-route web caching scheme is novelisation of the En-Route Caching proposed by [12] as an optimization for the file placement along the path from the cache or server to the client as it requires moderately more coordinating among the en route caches.
To model the caching and file placement of this scheme we are going to use the network

model in Figure 1.2 the graph is modeled as $G = (V; E)$,where V is the caches that contains an en-route cache and E is the network links and both server and client are attached to a node V.
We take for example one server only and clients requesting files contained in the server, when a request is made the request goes along the path to the server and is satisfied by the first cache node along the path that contains the requested file, the file is than transmitted down stream back to the client.

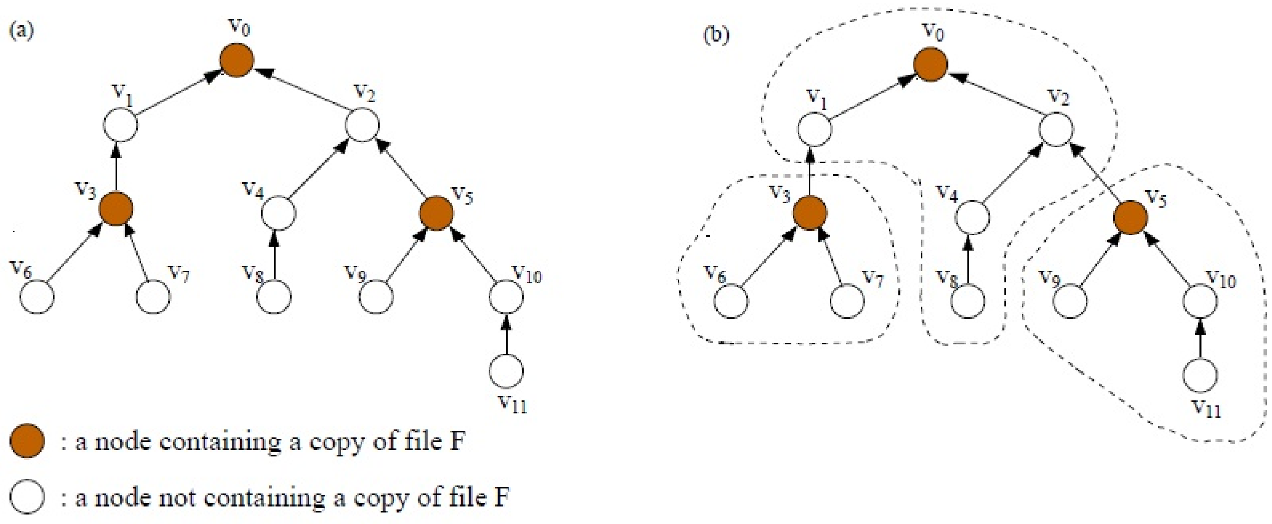

(a)
v0
v1
v2
v3
v4
v5
v6
v7
v8
v9
v10
v11
● : a node containing a copy of file F
○ : a node not containing a copy of file F
(b)

Figure 1.2: (a) En-Route Web Caching (b) Sub-trees Corresponding to Cached File Copies.[2]

Figure 1.2(a) is an a example for such tree topology where v_0 is the node associated with the server, while all other nodes are associated with en-route caches,every cache (or server)node that satisfied a request for any web file F, these set of nodes satisfies every request for the web file F from the sub-tree, in Figure1.2(b) there are three sub-trees where three nodes contains the requested web file F, in every sub-tree only one node contains the file F and its the node closest to the server.

1.3.1.2 Implicit schemes

These schemes uses small information exchange between cache routers to make the last decision about where to cache the content, these schemes doesn't need prerequisite information s such as cache network topology. example we take ProbCache.

ProbCache Every node cache probability varies because of the inversely corresponding to the distance from the cache to the node that made the request to model this scheme, [4] approached the problem from that caching capability point of view, ProbCache cache scheme is based on two value factors:

- TimesIn: is the content that a delivery path has to cache.
- CacheWeight: is the balancing factor that's based on TimesIn factor and the distance between the server and user

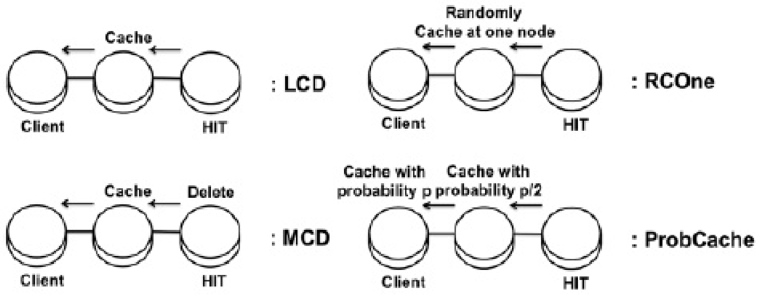


Figure 1.3: Iimplicit Coordination Approaches. [3]

Times In Factor To better understand it, we consider the two users in figure 1.4, the total cache capacity of a path is $\sum_i^n N_i$, from Figure1.4the distance between the server and $user1$ is 5 nodes $n_1 = 5$ for $request1$ and 4 nodes from $user2$ $n_2 = 4$ for $request2$ gray nodes represent the nodes that are going to be shared by both users and white nodes are the one are exclusive to user1 and black ones are exclusive for $user2$, TimesIn factor represent the number of times the path can afford to cache the file, TimesIn factor can be calculated as follows:

$$TimesIn(x) = \frac{\sum_{i=1}^{c=x-1} N_i}{T_{tw} N_x}$$

CacheWeight: To decide the number of times that the path can afford to cache the packet that TimesIn indicated we use the following formula:

$$CacheWeight(x) = \frac{x}{c}$$

where x is the of TSB the header and c is the TSI value.

ProbCache: Probabilistic In-Network Caching The product of TimesIn and CacheWeight is called ProbCache and it represents the caching probability of each node along the path from the server to the users depending depending on their TSI and TSB values.

$$ProbCache(x) = \frac{\sum_{i=1}^{c=x-1} N_i}{T_{tw} N_x} * \frac{x}{c}$$

[H] table of Model notation[4]

Symbol	Meaning
n	Number of caches on the path
N$_i$	Cache memory in r$_i$ that holds $1 - \text{sec}$ worth of traffic
T$_{tw}$	Target Time Window (set to 10 secs here)
TSI,	Time since Inception (Header field - Request Message):Hop-Distance from Client, Value range: 1 to n
TSB, x	Time since Birth (Header field - Content Message):Hop-Distance from Server, Value range: 1 to n

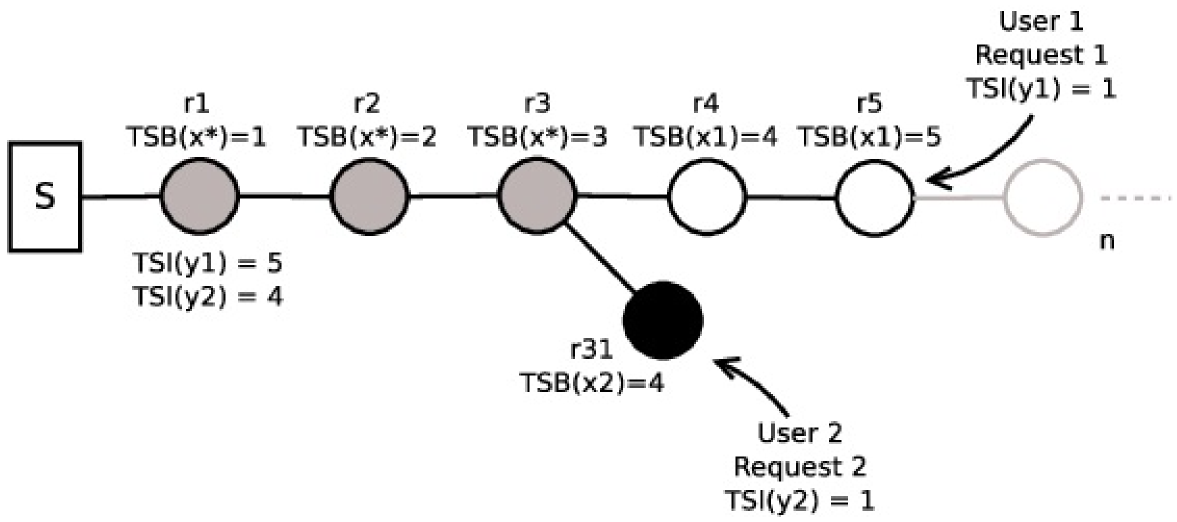


Figure 1.4: Design Topology [4]

1.3.2 probabilistic schemes

In this cache schemes, every node is independent and calculate its own feasibility to store a content,the decisions to cache and maintain their own policies and every node runs her canonical caching policy respectively which is usually based on historical usage or requesting frequency.

1.3.2.1 Leave Copy Everywhere (LCE) Caching Scheme

LCE is The default caching strategy in NDN [1]/CCN[2], leave copy everywhere as its name implies leaves a copy of the content all the way along the path from the server to user, to explain it more we take figure 1.5 from there we have the nodes from $R1$ to $R5$ that separates the user and server, when the request is sent from the user it traverse each node, from figure1.5 the request traverse each nodes until it get a hit than its copy's the content from the request from $R5$ to all the other nodes down the path to the user.

[1]Named Data Networking is an architectural realization of the broad Information Centric Networking (ICN) vision that enables communications by named, secured data at the network layer[13]

[2]Content Centric Networking: is a novel architecture that is shifting host-centric communication to a content-centric infrastructure[14].

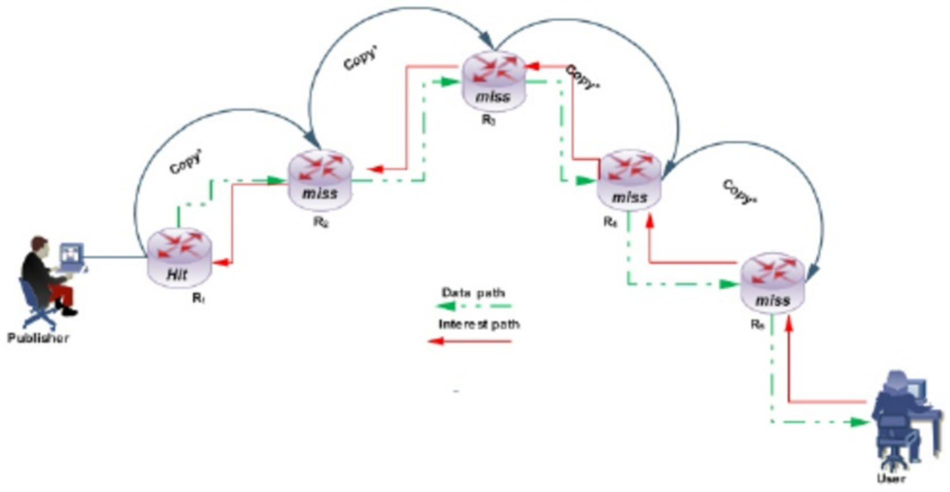


Figure 1.5: Leave copy everywhere

1.3.2.2 Age based Probabilistic Caching Scheme

To better understand it we are going to use the $LeaveCopyEverywhere(LCE)$ which is the original caching scheme of CCN that leaves a copy of the content in all the intermediate caches along the reverse path, this scheme increases the user quality of experience and content availability, by removing data redundancy and reducing delays resource inefficiency we can bypass this scheme only limitation, here [15] they proposed another type of (LCE), the age-based cooperative cache, in this type they added an age to every piece of content so that when it reach a certain age the content is automatically removed or replaced from the node cache, the age is assigned based on the popularity of the content within the network and the Distance between the node and the server, so the longer the distance and the more popular the content is the longer age it has.

1.4 Cache Replacement

When a cache node is chosen by the placement scheme to cache the new content the node checks if the content size is equal or higher than the content size. If its equal or higher than the content can be cached in that node if its lower than the node need to free some space for the new content to be cached. To free space the node need to decided what content to delete and replace by the new content, there are many algorithms that deals with this kind of situation and in this part we are going to talk about the three of them.

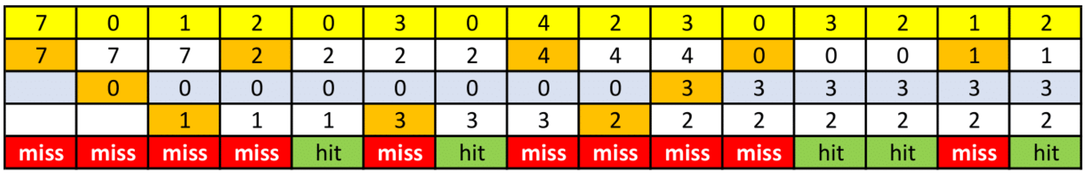
1.4.1 LRU(Least Recently Used)

LRU is a cache eviction policy which in case we have a full cache memory gives the ability to remove items in a certain order or restricted order.
The LRU cache policy removes the least recently used element of its order if the cache structure is over capacity. For us to be able to add a new element if an element inside the cache is accessed this element moves to the top of the list and becomes the most recently used as an example we are going to use a two dimension array with three lines that represent the size of our cache and a number of cases that represent the item we want

to store on the cache successively, when a new item is added to the cache and the item wasn't there already we call it a cache miss because the point of having cache is to find item we need on it not saving data on it so the more cache miss we find the less effective this cache replacement scheme is when you want to add an element but its already this is called a cache hit and the element becomes the most recently used item.
after testing the algorithm on the following list of numbers list of Numbers:

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2$$

we got the results expressed in Figure1.7.



7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3
		1	1	1	3	3	3	2	2	2	2	2	2	2
miss	miss	miss	miss	hit	miss	hit	miss	miss	miss	miss	hit	hit	miss	hit

Figure 1.6: Table representing an example of the LRU algorithm

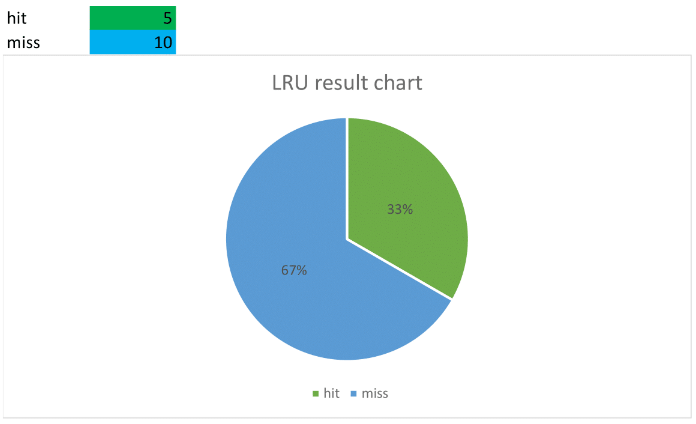


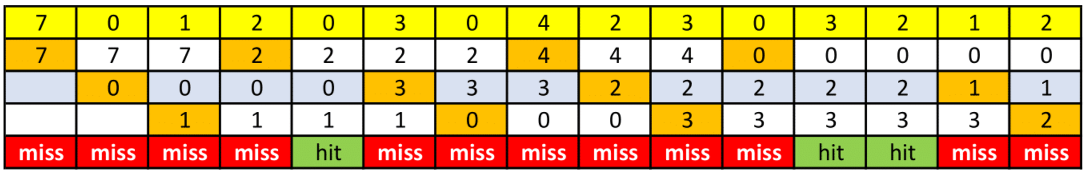
Figure 1.7: LRU example results chart

1.4.2 FIFO(First In First Out)

In this cache scheme to free space we will do as the name signify and it deleting the first item in which mean that the first item added to the table will be deleted the following example will present with the red the cache miss and the cache hits in green. after testing the algorithm on the following list of Numbers:

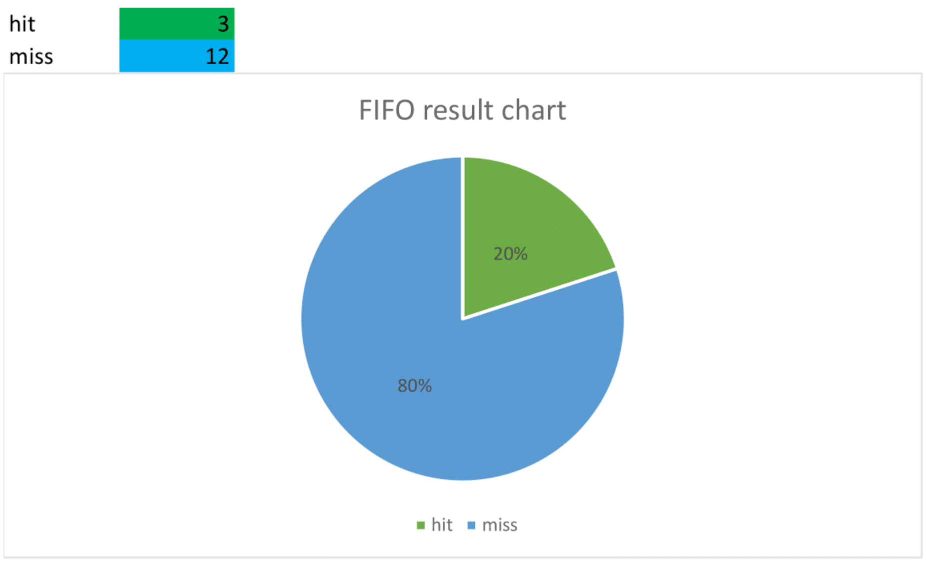
$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2$$

we got the results expressed in Figure1.9..



7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	0	0	0	0	3	3	3	2	2	2	2	2	1	1
		1	1	1	1	0	0	0	3	3	3	3	3	2
miss	miss	miss	miss	hit	miss	miss	miss	miss	miss	miss	hit	hit	miss	miss

Figure 1.8: Table representing an example of FIFO algorithm



hit	3
miss	12

Figure 1.9: FIFO example result chart

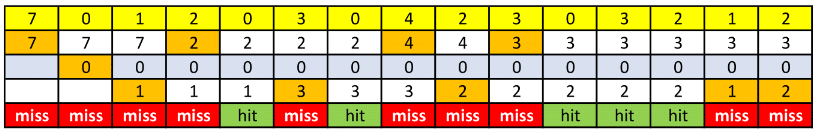
1.4.3 LFU(Least Frequently used)

In this algorithm each item cached in the node have frequency field, when a new item is added he item gets a frequency of zero and with each call or demand for the item the item frequency is increased by one. when the cache is full, the item with the least frequency is

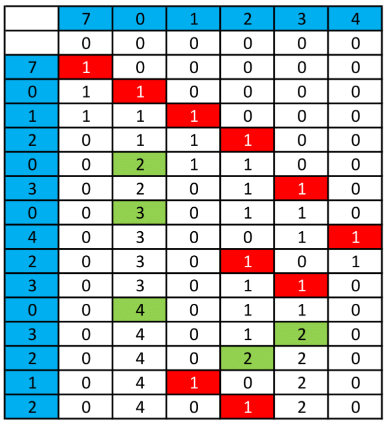
replaced, in case of two items having the same frequency the LFU algorithm uses either FIFO or LRU to choose which content to replace with new content which in its turn gets a frequency of zero.
after testing the algorithm on the following list of Numbers:

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2$$

We got the results expressed in Figure1.11.

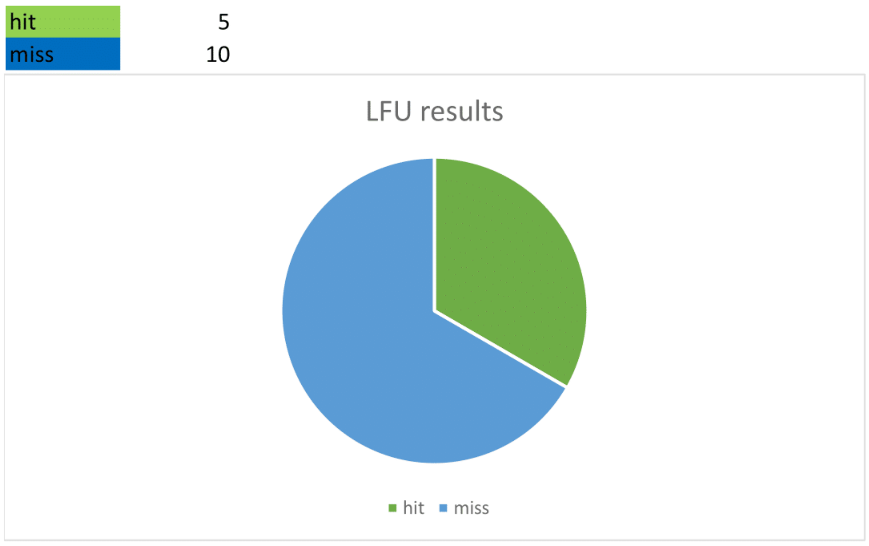


7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	7	7	2	2	2	2	4	4	3	3	3	3	3	3
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	1	2
miss	miss	miss	miss	hit	miss	hit	miss	miss	miss	hit	hit	hit	miss	miss



	7	0	1	2	3	4
	0	0	0	0	0	0
7	1	0	0	0	0	0
0	1	1	0	0	0	0
1	1	1	1	0	0	0
2	0	1	1	1	0	0
0	0	2	1	1	0	0
3	0	2	0	1	1	0
0	0	3	0	1	1	0
4	0	3	0	0	1	1
2	0	3	0	1	0	1
3	0	3	0	1	1	0
0	0	4	0	1	1	0
3	0	4	0	1	2	0
2	0	4	0	2	2	0
1	0	4	1	0	2	0
2	0	4	0	1	2	0

Figure 1.10: Tables representing the steps taken the LFU algorithm and the frequency changes during these steps



hit	5
miss	10

Figure 1.11: LFU example results chart

1.5 Conclusion

In this chapter, we talked about the different cache management schemes and deducted the difference between cache placement and replacement, we also mentioned some basic cache placement schemes examples.

Chapter 2

Data Mining

2.1 Introduction

In this chapter we are going to talk about Artificial intelligence and its types and branches, our focus will be directed to data mining and specially association itemsets mining precisely the apriori algorithm which we will explain its steps.

2.2 Artificial intelligence

Artificial intelligence (AI) is the science to make machines smart using algorithm to allow computers to solve problems which use to be solved only by humans, AI is a science with multiple approaches but made huge advancement in machine learning and deep learning, Among the fields of AI there is data mining field which is highly used in data extraction fields

2.3 Machine learning

Machine Learning is a sub field of artificial intelligence, it's a quite vast field that is expanding rapidly, being continually partitioned and sub-partitioned into different sub specialties and types of machine learning.
There are some basic common threads, however, and the overarching theme is best summed up by this oft-quoted statement made by Arthur Samuel way back in 1959: "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.“
"And more recently, in 1997, Tom Mitchell gave a "well-posed " definition that has proven more useful to engineering types"
"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."[16]

2.3.1 Types of machine learning

Machine Learning solves problems that cannot be solved by numerical means alone. Two of the most widely adopted machine learning methods are supervised ML and unsupervised ML, but there are also other methods of machine learning. Here's an overview of the most popular types

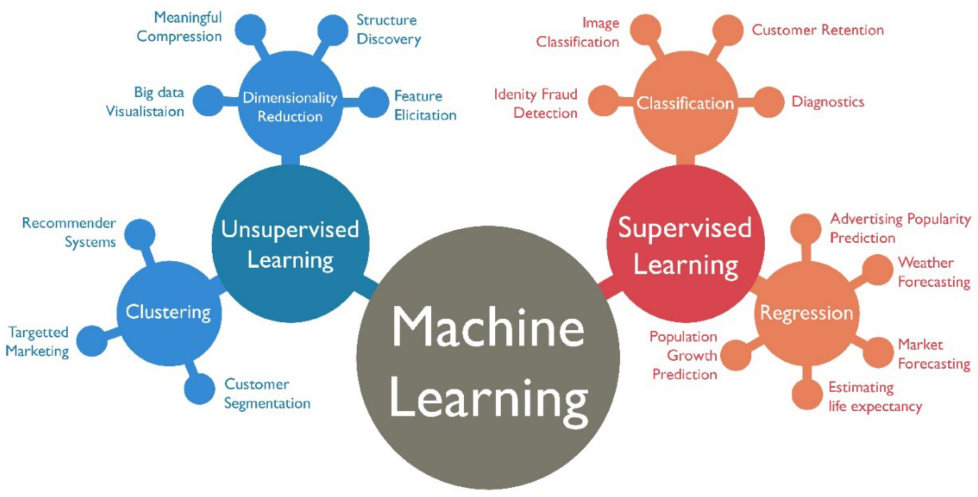


Figure 2.1: Machine Learning subcategories

2.3.1.1 Supervised machine learning

These algorithms are trained using labeled examples. For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression and prediction (check figure 1.2), supervised learning uses patterns to predict the values of the label on additional unlabeled data. Supervised learning is commonly used in applications where historical data predicts likely future events.

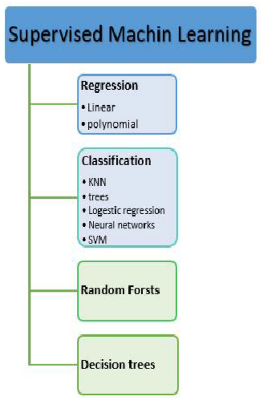


Figure 2.2: Algorithms of supervised machine learning

2.3.1.2 Unsupervised machine learning

This type of machine learning is used against data that has no historical labels. The system is not told the "right answer". The algorithm must figure out what is being shown. The goal is to explore the data and find some structure within. Unsupervised learning works well on transitional data. Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition. These algorithms are also used to segment text topics, recommend items and identify data outlets[17]

2.3.1.3 Semi-supervised machine learning

This subcategory is used for the same applications as supervised learning. But it uses both labeled and unlabeled data for training, typically a small amount of labeled data with a large amount of unlabeled data (because unlabeled data is less expensive and takes less effort to acquire). This type of learning can be used with methods such as classification, regression and prediction. Semi-supervised machine learning is useful when the cost associated with labeling is too high to allow for a fully labeled training process. Early examples of this include identifying a person's face on a webcam[17]

2.3.1.4 Reinforcement machine learning

It's often used for robotics, gaming and navigation. With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards. This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time. The agent will reach the goal much faster by following a good policy. So, the goal in reinforcement machine learning is to learn the best policy.[17]

2.3.2 Deep learning

Deep learning is a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks (check figure 2.1). In other words, it mirrors the functioning of our brains. Deep learning algorithms are similar too how nervous system structured where each neuron connected each other and passing information.

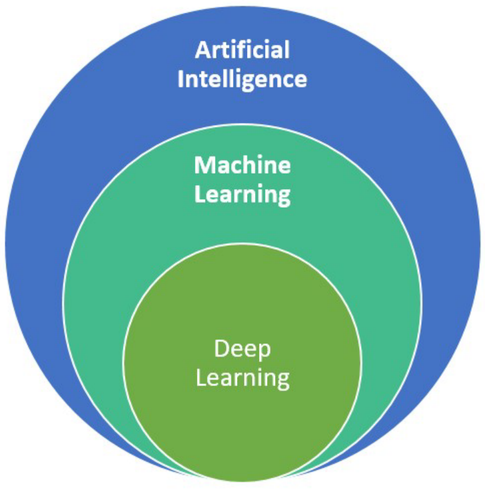


Figure 2.3: The relation between AI, machine learning and deep learning

2.4 Data mining tasks

Data Mining is the computer-assisted process of data sets sorting for pattern identification and relationship establishment, data mining solve problems through the extraction of knowledge from large amount of data and data sets through some data mining techniques.

”Data mining is the process of discovering interesting patterns and knowledge from large amounts of data. The data sources can include databases, data warehouses, the Web, other information repositories, or data that are streamed into the system dynamically.”[18]

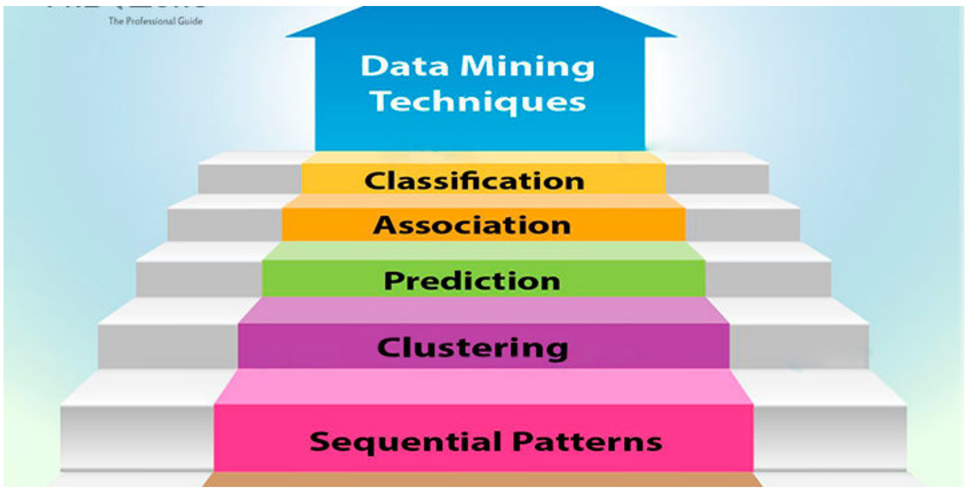


Figure 2.4: Data Mining Techniques

There are many techniques each has its own characteristics and goals depending on your source for example figure2.4. these techniques can be classified into two types:

Predictive.

- Classification
- Regression
- Time series Analysis
- Prediction

Descriptive:.

- clustering
- Summarization
- Association Rules
- Sequence Discovery

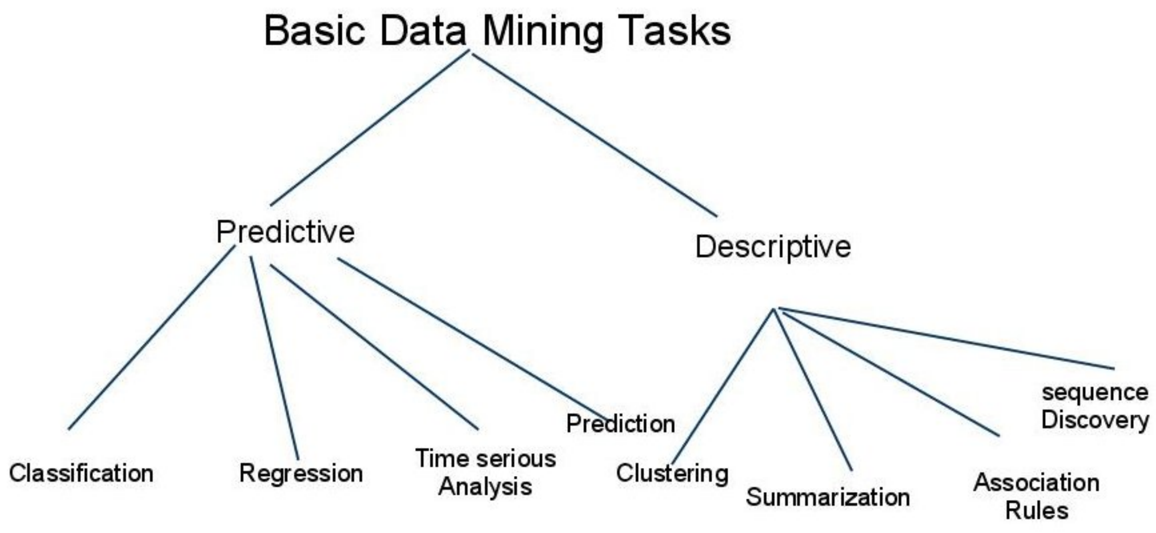


Figure 2.5: Data Mining Techniques Graph[5]

2.4.1 Classification Analysis

Classification is a data mining technique that assign a class or category to an item or a collection of items, the classification goal is to accurately predict for each target in a data set. it is also defined as " a process of assigning new entities to existing defined class by examining the entities features. Classification makes decision from unseen cases by building of past decisions"[6] .for example Classification Analysis can be used to organize houses into categories like crime size rate, view......

Requirements of the Classification Techniques. ”The basic requirements of classification techniques includes the construction of the model and the model usage. These requirements are defined explicitly”[6].

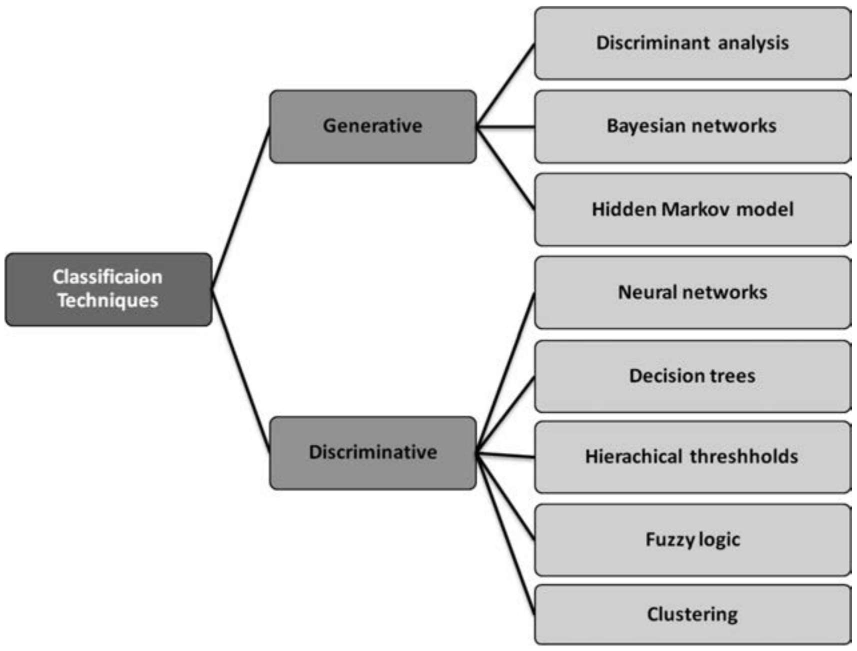
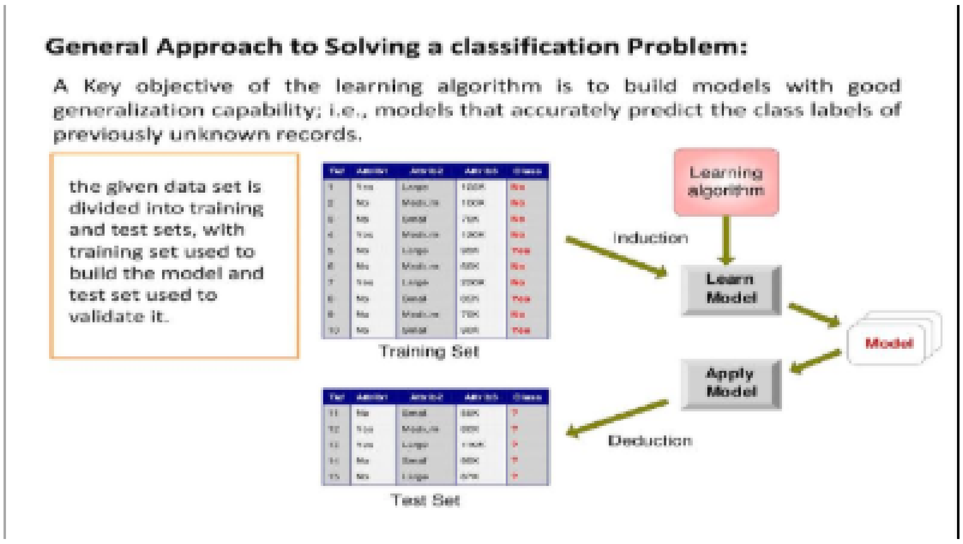


Figure 2.6: the Classification Techniques[6]

Construction of the model. ”Every sample of an object is assigned to a predefined class label. These objects or subset data are also known as training data set. Constructed models are always based on the training sets which represents as classification rule or decision trees. The building of models with good generalization capability is a key objective of the learning algorithm, for instance models that accurately predict the class labels of previously unknown records.”[6].
Figure2.7: illustrate the general approach to solving problems using classification.



General Approach to Solving a classification Problem:

A Key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previously unknown records.

the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Figure 2.7: General Approach to Solve Problem Using the Classification Technique[6]

Model Usage.

- Classification of unknown objects is performed based on the constructed model.
- Resultant class label compare with the class label of test sample.
- Calculate the percentage of test sample and accuracy of model should be compare with training sample.
- There are always differences between the test sample data and training sample data.

2.4.2 Regression Analysis

Regression is a data mining technique used for numeric values prediction of a given data set. ”Regression is a data mining (machine learning) technique used to fit an equation to a dataset.”[19] Given other values Regression can predict the value of a product or service, multiple industries use Regression Analysis for marketing planning, financial forecasting, environmental modeling and analysis of trends.

Types of Regression Techniques.

- Simple Linear Regression
- Standard multiple regression.
- Stepwise multiple regression.
- Hierarchical regression.
- Setwise regression.

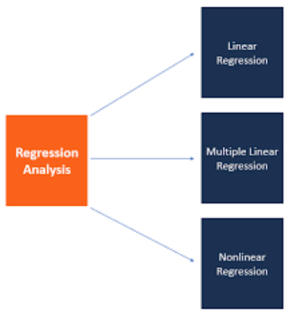


Figure 2.8: Regression techniques

2.4.3 Clustering

Clustering is the process of grouping a set of data objects that have high similarity into multiple groups or clusters, but are very dissimilar to objects in other clusters. The attribute values describing the objects are assessed based on dissimilarities and similarities of the attribute, values describing the objects and often involve distance measures. Clustering as a data mining tool has its roots in many application areas such as biology, security, business intelligence, and Web search.[18].

2.4.3.0.1 Requirements for Cluster Analysis these are some of the requirements for clustering as a data mining tool,

- **Scalability:**Highly scalable clustering algorithms are needed in order to avoid biased results from clustering sample of a given large data, clustering algorithms work well on small data sets containing fewer than several hundred data objects. [18]
- **Ability to deal with different types of attributes:**Clustering all types of data types became a requirement to clustering algorithm who are initially assigned to cluster numeric (interval-based) data. since applications are more and more improving new complex data types appeared thee for an all type clustering algorithm is a must.[18]
- **Discovery of clusters with arbitrary shape:**The basic methods tends to find spherical clusters, how ever in many cases and phenomena clustering to find the frontier are not spherical and there for. It is important to develop algorithms that can detect clusters of arbitrary shape.[18]
- **Requirements for domain knowledge to determine input parameters:**The quality of clustering difficult to control when the Parameters are hard to determine specially for high-dimensionality data sets where users have yet to grasp a deep understanding of their data.[18]
- **Ability to deal with noisy data:**Clustering methods can be sensitive to such noise and may produce poor-quality clusters, specially that most of the real-world data sets contain outliers and/or missing, unknown, or erroneous data. Therefore, we need clustering methods that are robust to noise.[18]

Clustring Techniques. The major fundamental clustering methods can be classified into the following categories:

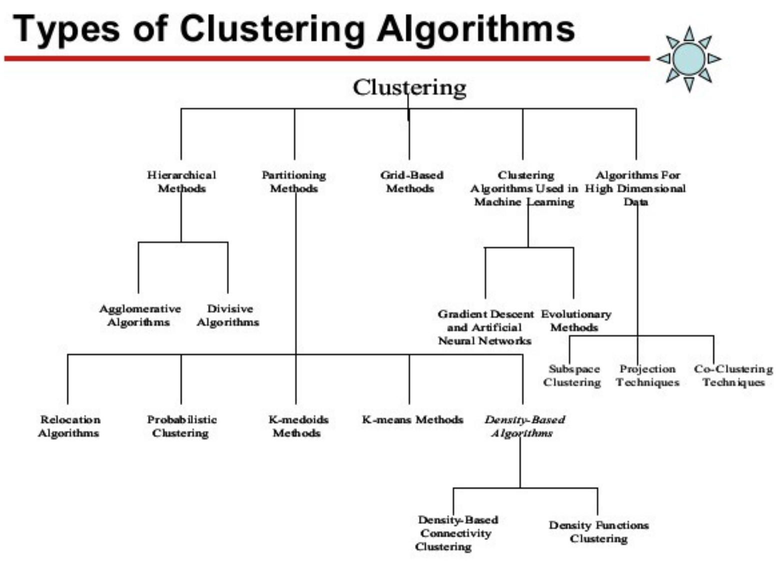


Figure 2.9: the major fundamental clustering techniques[7]

2.4.4 Association rules

Association Rule learning is a rule based data mining technique,this technique attempts to find interesting associations and relationships hidden in large data-sets,the most commune example of Association itemsets mining is the market basket analysis where they used the clients item baskets to determine the items frequently bought together, with a set of transaction we can find itemsets that can predict the occurrence of an item based on the occurrence of another item.

The main thought around the Association Rule mining is the IF-THEN relationship, where IF an item A is chosen than the chances of an item B is chosen are higher.

Figure 2.10: IF-THEN relationship

There are two elements of these itemsets:

1. Antecedent (IF): This is an item/group of items that are typically found in the Itemsets or Datasets.
2. Consequent (THEN): This comes along as an item with an Antecedent/group of Antecedents.

There are many algorithms that can implement association rule mining,one of them is Apriori algorithm.

2.5 Apriori algorithm

The Apriori algorithm was introduced for the first time in 1994 by [20] and got its name due to the use of prior knowledge of frequent item-set.

"Apriori algorithm is a data mining method which outputs all frequent item-sets and association itemsets from given data."[21]

The apriori is a classic algorithm useful for mining frequent item-sets and relevant association itemsets, it operates on data-sets containing large numbers of transactions where transactions are the items bought or requested by a user successively.

The apriori algorithm helps client ease their item search processes and sales performance of the departmental store, in the healthcare and medical field the apriori algorithm can detect the adverse drug reactions (ADR) by indicating combination of medications and patient characteristics that could lead to ADR.

2.5.1 Theory behind the Apriori algorithm

Let Σ be an alphabet of n item I$\subseteq T_i \rightarrow$ I=$\{a_1, a_2\}$
Σ= $\{a_1, a_2, a_3..a_n\}$
A transaction is an itemset of Σ
$T \subseteq \Sigma$ $T_i \rightarrow$I=$\{a_1, a_2, a_3\}$
An itemset I is a subset of T_i

I$\subseteq T_i \rightarrow$ I=$\{a_1, a_2\}$
A transaction Dataset D is a set of n transactions and N is the number of transactions in D
D=$\{T_1, T_2, T_3, T_4, ...T_n\}$ $\rightarrow N =\mid D \mid$
The Apriori algorithm has three significant components that gives it strength, these components are as follow:

- Support
- Confidence
- Lift

For example in the a supermarket lets suppose we have 4000 transaction and we want to find the Support, Confidence, and Lift for two items lets say bread and jam because people tend to buy this two item together frequently.

lets say that out of 4000 transaction 400 contains jam an 500 contains bread, from those numbers there 100 transaction containing both jam and bread, using this data we can calculate the Support, Confidence, and Lift of these two items using these formulas:

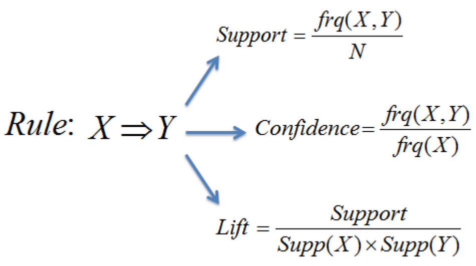

Support = frq(X,Y) / N
Rule: X ⇒ Y
Confidence = frq(X,Y) / frq(X)
Lift = Support / Supp(X) × Supp(Y)

Figure 2.11: Basic terminologies

2.5.1.1 Support

Support refers to the default popularity of any item,support is calculated as the quotient of transactions containing a particular item divided by the total number of transactions to calculate the support of an item B we go as follow:

$$Support(B) = \frac{(Transactions\_containing(B))}{(Total\_Transactions)}$$

to make a better example we are going to calculate the support of jam out of 4000 transaction

$$Support(jam) = \frac{(Transactions\_containing\_jam)}{(Total\_Transactions)}$$

$$Support(jam) = \frac{400}{4000}$$

$$Support(jam) = 10\%$$

2.5.1.2 Confidence

Confidence is the likelihood that an item B is also bought when an item A is bought, in our example of bread and jam there is more likely chance to buy beard when you buy jam, to mathematically calculate it we divide the number of transactions containing both A and B items by the total number of transaction containing A:

$$Confidence(A \to B) = \frac{(Transactions\_containingboth(AandB))}{(Transactions\_containingA)}$$

Coming back to our to our example we have 100 transaction containing both jam and bread and 400 transaction containing only jam, by dividing these two numbers we get the likelihood of buying bread when jam is bought:

$$Confidence(jam \to bread) = \frac{(Transactions\_containing\_both(jam\_and\_bread))}{(Transactions\_containing\_jam)}$$

$$Confidence(jam \to bread) = \frac{100}{400}$$

$$= 25\%$$

2.5.1.3 Lift

Lift is the increase in the ratio of the sale of an item B when A is sold it is also a measure of how A an B are really related to each other rather than happening accidentally , lift can be calculated by dividing the the confidence of between an item A and B by the support of an item B:

$$Lift(A \to B) = \frac{(Confidence(A \to B))}{(Support(B))}$$

Coming back to our to our example of bread and jam we have a the confidence between jam and bread and the support of bread and by these we get the following result:

$$Lift(jam \to Bread) = \frac{(Confidence(jam \to Bread))}{(Support(Bread))}$$

$$Lift(jam \to Bread) = \frac{25}{12.5}$$

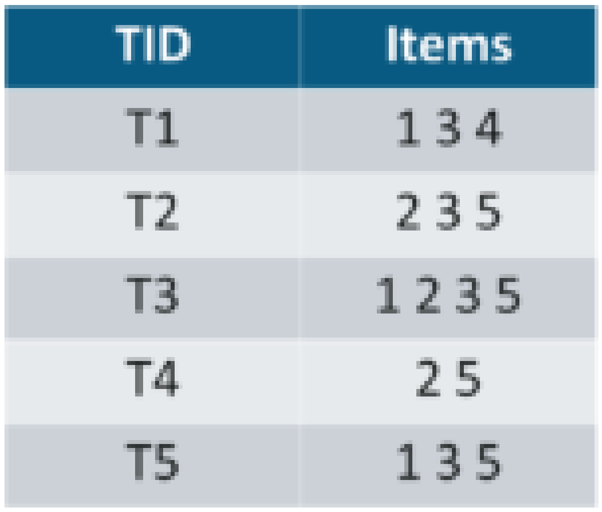
$$Lift(jam \to Bread) = 2$$

lift result explains the strength of a rule and these results are separated in three kind of result lift=1 and lift>1 and lift<1 each result has its representation as follow:

- lift=1: means that the article And B are statically independent
- lift>1: means that the article A and B have a strong relation ship between them and the lift value means that if u buy the item A there lift chances of buying the item B in our example we buy jam than the chances of buying Bread is raised to two times 2
- lift<1: means that there is no relationship between the two articles and it is unlikely to buy item A if item B is bought.

2.5.2 Apriori algorithm steps

Apriori algorithm uses frequent itemsets to generate association itemsets,the basic concept of this algorithm is that every subset of a frequent itemset must be frequenht itemset, a frequent itemset is an item set that has support value higher than the minimum support value.
To explain the algorithm in much sufficient way we are going use the following that dataset as an example:

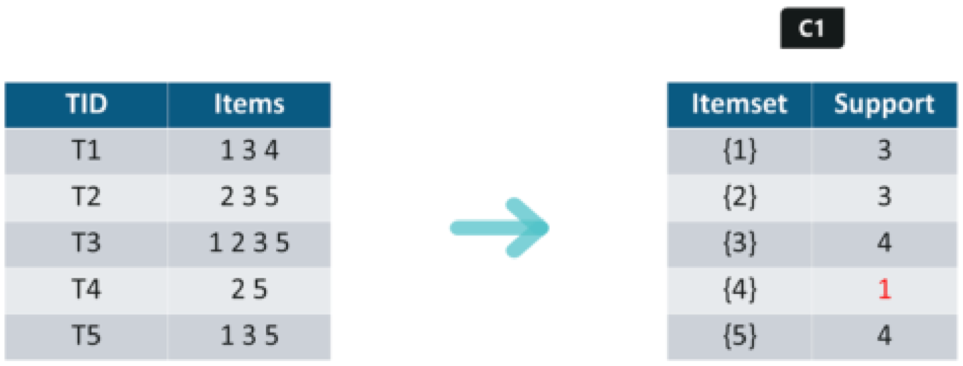


TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

Figure 2.12: Example DATASET

step 1

- **Iteration 1:** for this iteration we are going set our minimal support value to 2 and create the list of item set of size 1 and calculate their support:



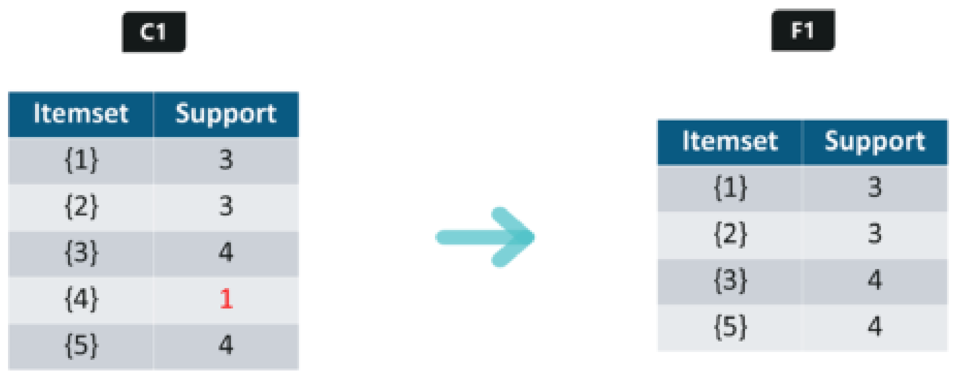
TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

C1

Itemset	Support
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4

Figure 2.13: Iteration 2: sets of item of size 1

as we notice from Figure:2.13 the support value of the itemset 4 is less than the minimal support value, so the next step is to remove this itemset from the table.



C1

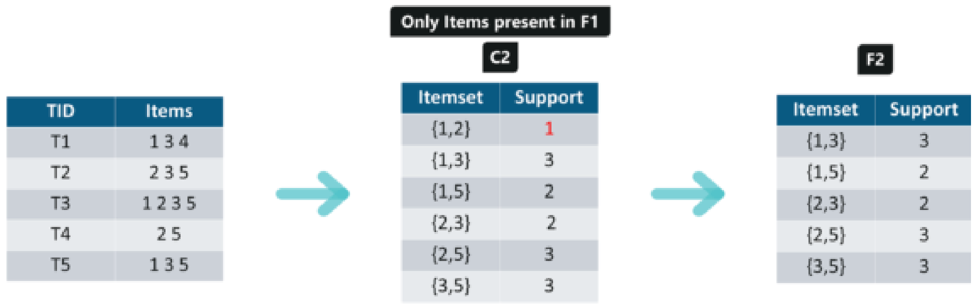
Itemset	Support
{1}	3
{2}	3
{3}	4
{4}	1
{5}	4

F1

Itemset	Support
{1}	3
{2}	3
{3}	4
{5}	4

Figure 2.14: Table C1 after deleting the itemsset4

- **Iteration 2:** for this iteration we are going create the list of item set of size 2 and calculate their support, All the items set combinations of F1 are used in this iteration.



TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

Only Items present in F1

C2

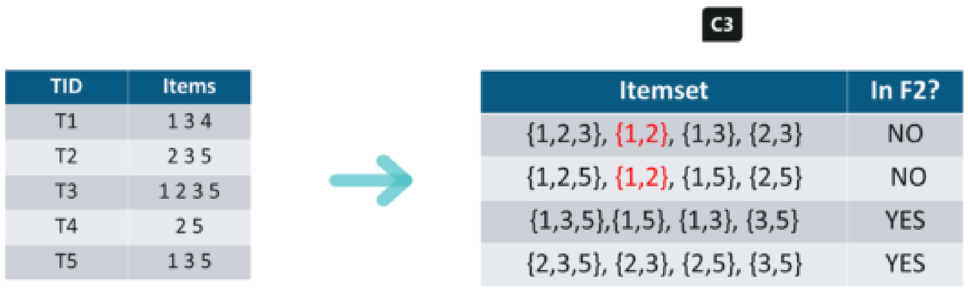
Itemset	Support
{1,2}	1
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

F2

Itemset	Support
{1,3}	3
{1,5}	2
{2,3}	2
{2,5}	3
{3,5}	3

Figure 2.15: Creation of item sets of size 2 from the combinations in F1

The itemsets that have support values less than 2 are removed and in our case the item set 1,2 as in figure 2.15, the next step is the pruning and its the part that give the apriori algorithm its strength and makes it one of the best algorithm for association rule mining **pruning:**the pruning is dividing/// the itemset into subsets than eliminating the subsets that have support values less than 2



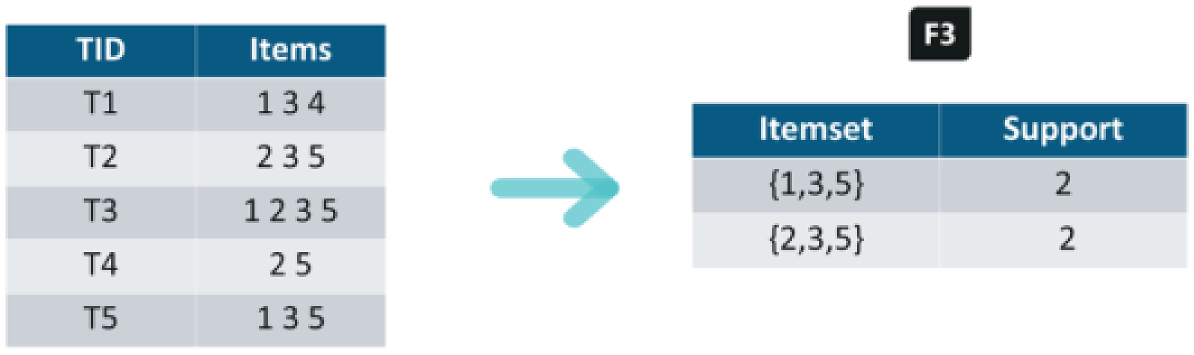
TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

C3

Itemset	In F2?
{1,2,3}, {1,2}, {1,3}, {2,3}	NO
{1,2,5}, {1,2}, {1,5}, {2,5}	NO
{1,3,5},{1,5}, {1,3}, {3,5}	YES
{2,3,5}, {2,3}, {2,5}, {3,5}	YES

Figure 2.16: Subsets of the itemset

Iteration 3:1,2,3 and 1,2,5 are discarded as they both contain 1,2, this iteration is considered the high light of the apriori algorithm



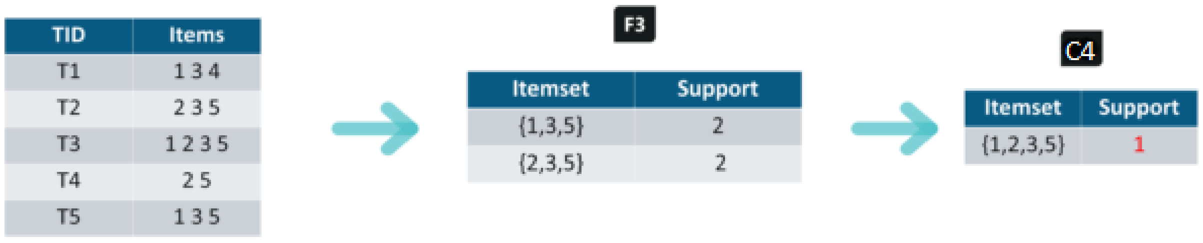
TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

F3

Itemset	Support
{1,3,5}	2
{2,3,5}	2

Figure 2.17: Item sets after iteration 3

Iteration 4: Using sets of F3 we will create C4.



TID	Items
T1	1 3 4
T2	2 3 5
T3	1 2 3 5
T4	2 5
T5	1 3 5

F3

Itemset	Support
{1,3,5}	2
{2,3,5}	2

C4

Itemset	Support
{1,2,3,5}	1

Figure 2.18: Item sets after iteration 3

Since the Support of this itemset is less than 2, we will stop here and the final itemset we will have is F3.
from F3 we are going to extract the following itemsets:
For I = 1,3,5, subsets are 1,3, 1,5, 3,5, 1, 3, 5
For I = 2,3,5, subsets are 2,3, 2,5, 3,5, 2, 3, 5

step 2 Applying and creating itemsets: now we are going to create itemsets and apply them on the F3 itemsets and define the minimum confidence to 60For every subsets S of I, you output the rule

- S -> (I-S) (means S recommends I-S)
- if **support(I) / support(S) >= min_conf value**

From1,3,5
Rule 1: 1,3 -> (1,3,5 — 1,3) means 1 & 3 -> 5
Confidence = support(1,3,5)/support(1,3) = 2/3 = **66.66 > 60**

Rule 1 is **Selected**

Rule 2: 1,5 -> (1,3,5 — 1,5) means 1 5 -> 3
Confidence = support(1,3,5)/support(1,5) = 2/2 = **100 > 60**
Rule 3 is **Selected**

Rule 4: 1 -> (1,3,5 — 1) means 1 -> 3 5
Confidence = support(1,3,5)/support(1) = 2/3 = **66.66 > 60**
Rule 4 is **Selected**

Rule 5: 3 -> (1,3,5 — 3) means 3 -> 1 5
Confidence = support(1,3,5)/support(3) = 2/4 = **50 <60**
Rule 5 is **Rejected**

Rule 6: 5 -> (1,3,5 — 5) means 5 -> 1 3
Confidence = support(1,3,5)/support(5) = 2/4 = **50 < 60**
Rule 6 is **Rejected**

From2,3,5
Rule 1: 2,3 -> (2,3,5 — 2,3) means 2 3 -> 5
Confidence = support(2,3,5)/support(2,3) =2/2 = **100 > 60**
Rule 1 is **Selected**

Rule 2: 2,5 -> (2,3,5 — 2,5) means 2 5 -> 3
Confidence = support(2,3,5)/support(2,5) = 2/3 = **66.66 > 60**
Rule 2 is **Selected**
Rule 3: 3,5 -> (2,3,5 — 3,5) means 3 5 -> 2
Confidence = support(2,3,5)/support(3,5) = 2/3 = **66.66 > 60**

Rule 3 is **Selected**

Rule 4: 2 -> (2,3,5 — 2) means 1 -> 3 5
Confidence = support(2,3,5)/support(2) = 2/3 = **66.66 > 60**
Rule 4 is **Selected**

Rule 5: 3 -> (2,3,5 — 3) means 3 -> 2 5
Confidence = support(2,3,5)/support(3) = 2/4 = **50 <60**
Rule 5 is **Rejected**

Rule 6: 5 -> (2,3,5 — 5) means 5 -> 2 3
Confidence = support(2,3,5)/support(5) = 2/4 = **50 < 60**
Rule 6 is **Rejected**

step 3 the third and last step we are going to calculate the lift and define a minimum lift of 1, lift is calculate the following way:

- S -> (I-S) (means S relate I-S)

- if **support(I-S) /support(I)* support(S) >= min_lift value**

2.6 Conclusion

In this chapter we talked and explained some of the most important part of the Artificial intelligence, we talked about it's different branches and focused on the data mining branch specially the association rule mining, and also talked and explained the apriori algorithm which we are going use later on this paper to extract some itemsets out of a dataset.

Chapter 3

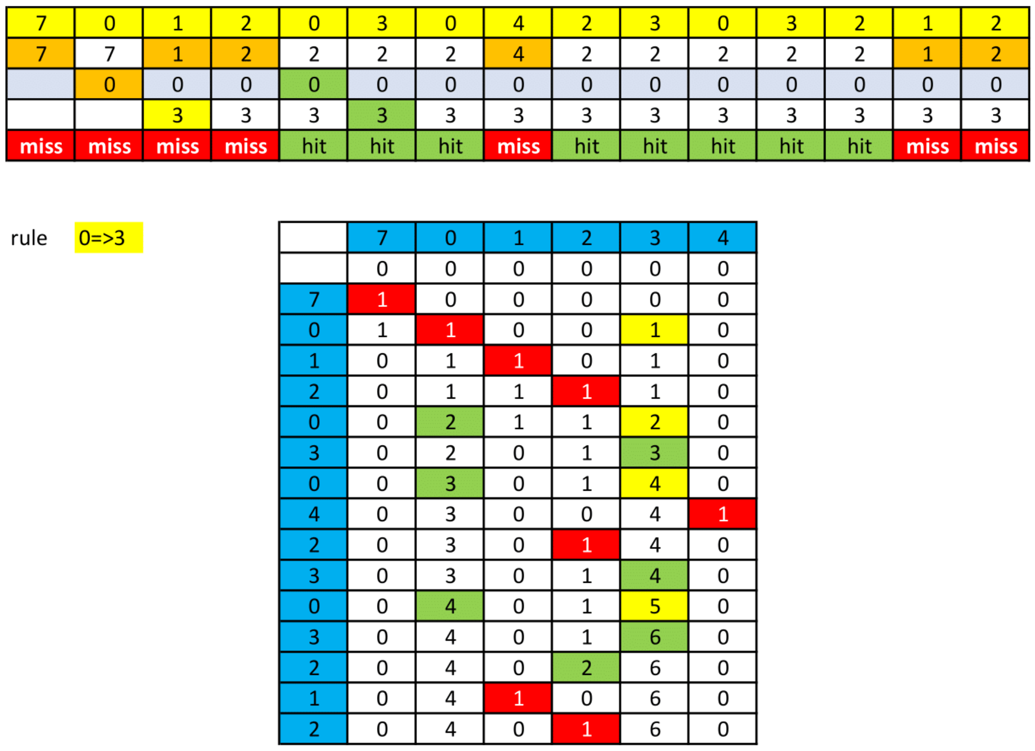
Design and implementation

3.1 Introduction

In this chapter we are going to follow the steps to implement the apriori algorithm and extract itemsets from a dataset, than we are going to use the itemsets extracted from the dataset to improve the results of the LFU algorithm and than do some tests in this implementation to compare between the LFU before and after adding the itemsets.

3.2 Improving LFU using apriori algorithm :

After extracting the itemsets out of the given dataset which represent the item use frequency habits of users (in this case the items are sold to customers by a French retail store), to better understand it we are going to use a list of numbers that we want to successively store in the cache numbers and a cache size of 3 as represented in the first table 1 in 3.1.
We also have a dictionary that contain the itemsets extracted by the apriori algorithm which we need for this study and in this example we are going to use only one rule due to the size of the dataset and just as an example. In the second table, the rows are the variant numbers contained on the list and the lines are the list. this table is used to follow up the change on the frequency of each item while we put the list into the cache, our work here was if an item about to be **put()** in the cache is present in the dictionary as key. We also add (**put()**) its value with it, if the value is not in the cache than it will be added and its frequency is set to one if not than its frequency will increase.
After testing the algorithm on the same list as the one we used for the LFU and LRU and FIFO algorithms in section 1.4 we got the following results as the table 1.10 shows, the amount of hit increased compared to the other algorithms while the miss rate decreased the results are expressed in chart3.2.



7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
7	7	1	2	2	2	2	4	2	2	2	2	2	1	2
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		3	3	3	3	3	3	3	3	3	3	3	3	3
miss	miss	miss	miss	hit	hit	hit	miss	hit	hit	hit	hit	hit	miss	miss

rule	0=>3

	7	0	1	2	3	4
	0	0	0	0	0	0
7	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	1	1	0	1	0
2	0	1	1	1	1	0
0	0	2	1	1	2	0
3	0	2	0	1	3	0
0	0	3	0	1	4	0
4	0	3	0	0	4	1
2	0	3	0	1	4	0
3	0	3	0	1	4	0
0	0	4	0	1	5	0
3	0	4	0	1	6	0
2	0	4	0	2	6	0
1	0	4	1	0	6	0
2	0	4	0	1	6	0

Figure 3.1: LFU with itemsets example

hit	8
miss	7

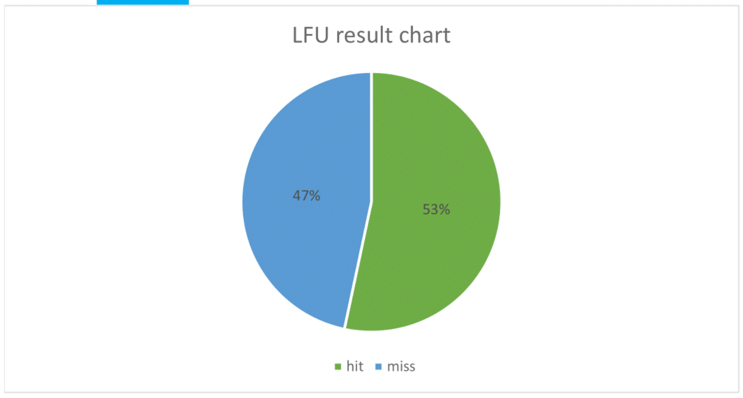


Figure 3.2: LFU with itemsets result chart

3.3 Softwares and tools

3.3.1 Python

Python [1] is an interpreted, object-oriented, high-level programming language with dynamic semantics. is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed [22].

3.3.2 Hardware

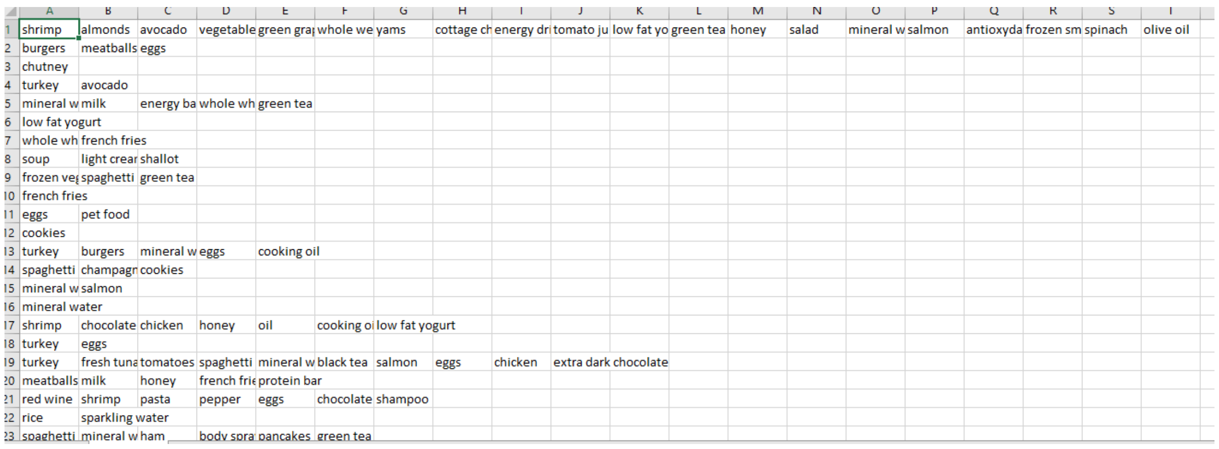
the following table represent the hardware materiel used for this implementation.

CPU	i5-4200H (2.8Ghz)
GPU	Nvedia GTX850M 4 Gb
Ram	8GB
OS	Windows 8.1 Pro

Table 3.1: The hardware used to run the tests

3.3.3 Dataset

the dataset [2] used in this thesis is 7500 transaction dataset of over a week at a French retail store, here is a screen shot of the dataset:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	shrimp	almonds	avocado	vegetable	green gra	whole we	yams	cottage ch	energy dri	tomato ju	low fat yo	green tea	honey	salad	mineral w	salmon	antioxyda	frozen sm	spinach	olive oil
2	burgers	meatballs	eggs																	
3	chutney																			
4	turkey	avocado																		
5	mineral w	milk	energy ba	whole wh	green tea															
6	low fat yogurt																			
7	whole wh	french fries																		
8	soup	light crear	shallot																	
9	frozen veg	spaghetti	green tea																	
10	french fries																			
11	eggs	pet food																		
12	cookies																			
13	turkey	burgers	mineral w	eggs	cooking oil															
14	spaghetti	champagn	cookies																	
15	mineral w	salmon																		
16	mineral water																			
17	shrimp	chocolate	chicken	honey	oil	cooking oi	low fat yogurt													
18	turkey	eggs																		
19	turkey	fresh tuna	tomatoes	spaghetti	mineral w	black tea	salmon	eggs	chicken	extra dark chocolate										
20	meatballs	milk	honey	french frie	protein bar															
21	red wine	shrimp	pasta	pepper	eggs	chocolate	shampoo													
22	rice	sparkling water																		
23	spaghetti	mineral w	ham	body spra	pancakes	green tea														

Figure 3.3: Screen shot of the dataset

[1]https://www.python.org/doc/essays/blurb/

[2]https://drive.google.com/file/d/1y5DYn0dGoSbC22xowBq2d4po6h1JxcTQ/view?usp=sharing

3.4 Implementation of the apriori algorithm

The following steps were used to implement the apriori algorithm:

3.4.1 Importing the libraries

First we downloaded the libraries we need for this implementation, the libraries are: apyori which is the main library for this implementation. There is also pandas to manipulate and analyse the dataset and numpy to manipulate mathematical fonctions and to Provide a MATLAB-like plotting framework of our dataset

```
 import numpy as np
 import matplotlib.pyplot as plt
 import pandas as pd
 from apyori import apriori
```

Listing 3.1: Importing the libraries

3.4.2 Importing the Dataset

Now we import the dataset by using pandas like the following:

```
 store_data = pd.read_csv('store_data.csv', header=None)
```

to have a look at it we call this function:

```
s tore_data.head()
```

Each row the dataset represent a transaction and each column represents an item, the NaN means that the item wasn't bought in that transaction. Since there is no header in this dataset the first row will be the header by default and to get rid of this problem we use the following command

```
header=None
```

3.4.3 Pre-processing the Dataset

In this phase we convert our pandas data-frame into a list of lists, because the Apriori library requires the dataset to be in form of list of lists, where each transaction in an inner list of the large list which is the new form of the dataset:

```
records = []
for i in range(0, 7501):
    records.append([str(store_data.values[i,j]) for j in range(0, 20)])
```

Listing 3.2: Pre-processing the Dataset

3.4.4 Applying Apriori

In this step we use the apriori class that we imported from the apyori library to apply the apriori algorithm to our dataset.
The apriori class requires some parameters to work, the first parameter is the list which is the new form of our dataset who we are going to extract the itemsets from, second parameter is *min_support* which is used to select the items with higher support value than the parameter the same way goes for the *min_confidence* and its selecting the item

with greater confidence value than the parameter, and we use *min_lift* to specify the minimum value of lift for the short list left of rule, the last parameter is *min_length* which specify the number of items u want in your itemsets, we use it as follow:

```
association_itemsets = apriori(records, min_support=0.0045,
 min_confidence=0.2, min_lift=3, min_length=2)
association_results = list(association_itemsets)
```

Listing 3.3: Applying Apriori

The second line convert the result of the function to a list for easier view and manipulation

3.4.5 Viewing the Results

The following script represent the numbers of itemsets got from the apriori algorithm where each item represent a rule:

```
print(len(association_itemsets))
```

This past code should result a 48 item means 48 rule, and in the next script we will show the first item in the association_itemsets list:

```
print(association_itemsets[0])
```

The result should be the following:

```
RelationRecord(items=frozenset({'light cream', 'chicken'}), support
 =0.004532728969470737, ordered_statistics[OrderedStatistic(items_base
 =frozenset({'light cream'}), items_add=frozenset({'chicken'}),
 confidence=0.2905982905982905 7, lift=4.84395061728395)])
```

Listing 3.4: Viewing the Results

3.5 Implementing LFU algorithm

This are the following steps too to implement the LFU caching scheme algorithm

3.5.1 Creating the data structure

The data structure we are using for this implementation is a list of dictionaries, the function to create this data structure takes one argument which is the capacity of the cache used for this implementation, this function create three dictionaries the first one is cache dictionaries which contains the items added or the values the second one is the count dictionary which represent the count of each item or value or as we called it before the frequency of each item in the cache, the third dictionary is the grouping dictionary to keep track of the grouping of the item on the list where the items are grouped to make it easy to extract the least frequent item of the cache

```
def __init__(self, capacity: int):
        self.cache = dict()
        self.countDict = dict()
        self.grouping = dict()
        self.capacity = capacity
```

Listing 3.5: Creating the data structure

3.5.2 The get() function

In short the **get()** function is a function to get an item from the cache, when the function get is called if the element is in the cache the function **get()** return the item and raise his frequency value by 1 if it is not in the cache the function returns -1

```python
def get(self, key: int) -> int:
  if key in self.cache:
    currentCount = self.countDict[key]
    self.countDict[key] = currentCount+1
    self.grouping[currentCount].remove(key)
  if len(self.grouping[currentCount]) == 0:
    del self.grouping[currentCount]
  if currentCount+1 in self.grouping:
    self.grouping[currentCount+1].append(key)
  else:
    self.grouping[currentCount+1] = [key]
    return self.cache[key]
  else:
    return -1
```

Listing 3.6: The get() function

3.5.3 The put() function

The **Put()** function is the one responsible of filling the cache where when an item is not found in the cache memory the Put() function add the item to the cache and give it a frequency of 1 if the cache is full the **Put()** function removes the item with the less frequency, if the item is already exist his frequency is increased by 1

```python
def put(self, key: int, value: int) -> None:

  if self.capacity <= 0:
  return
  if key in self.cache:

    self.cache[key] = value
    currentCount = self.countDict[key]
    self.countDict[key] = currentCount+1
    self.grouping[currentCount].remove(key)
  if len(self.grouping[currentCount]) == 0:
    del self.grouping[currentCount]
  if currentCount+1 in self.grouping:
    self.grouping[currentCount+1].append(key)
  else:
    self.grouping[currentCount+1] = [key]
  else:

  if len(self.countDict) >= self.capacity:
    lfuCount = min(self.grouping.keys())
    lfuKey = self.grouping[lfuCount].pop(0)
    print(self.grouping[lfuCount])

  if len(self.grouping[lfuCount]) == 0:
    del self.grouping[lfuCount]
    del self.cache[lfuKey]
    del self.countDict[lfuKey]
    self.cache[key] = value
  if 1 in self.grouping:
    self.grouping[1].append(key)
  else:
    self.grouping[1] = [key]
    self.countDict[key] = 1
```

Listing 3.7: The put() function

3.5.4 Compilation

First of all we created the object Lfucache and gave it a size of 24 item which is 20% of the items used in these dataset transactions

```
  lfuCache = LFUCache(24)
i teration=1000
```

Listing 3.8: Compilation

Than we made a **for** loop that will move the variable i from zero to the size our dataset list while using **Put()** to fill the cache, when the item is found a variable names hit will increase and when the its not found a variable names miss is incremented, when the loop is over we print out the result which are the content of the cache and the number of **hit** and **miss** we got along the way:

```
hit=0
miss=0
  for i in range(iteration):
    if store_data[i] in lfuCache.cache:
      hit=hit+1
    else:
      lfuCache.put(store_data[i],store_data[i])
      miss=miss+1
print("hit rate:",hit)
print("miss rate:",miss)
```

3.5.5 Results

The results should be the following where **Cache** contain the keys and values inside the cache and **Count Dict** contain each item and his frequency and **Grouping Dict** contains grouped items cache based on their frequency:

```
Capacity: 36
Cache: {'avocado': 'avocado', 'low fat yogurt': 'low fat yogurt', 'green
    tea': 'green tea', 'honey': 'honey', 'mineral water': 'mineral water
   ', 'spaghetti': 'spaghetti', 'salmon': 'salmon', 'olive oil': 'olive
   oil', 'burgers': 'burgers', 'meatballs': 'meatballs', 'eggs': 'eggs',
    'turkey': 'turkey', 'milk': 'milk', 'french fries': 'french fries',
   'chicken': 'chicken', 'frozen vegetables': 'frozen vegetables', '
   cooking oil': 'cooking oil', 'cookies': 'cookies', 'shrimp': 'shrimp
   ', 'pasta': 'pasta', 'chocolate': 'chocolate', 'ground beef': 'ground
    beef', 'energy bar': 'energy bar', 'escalope': 'escalope', 'mushroom
    cream sauce': 'mushroom cream sauce', 'soup': 'soup', 'hot dogs': '
   hot dogs', 'brownies': 'brownies', 'muffins': 'muffins', 'fresh tuna
   ': 'fresh tuna', 'energy drink': 'energy drink', 'herb & pepper': '
   herb & pepper', 'champagne': 'champagne', 'ham': 'ham', 'pancakes': '
   pancakes', 'yogurt cake': 'yogurt cake'}
```

```
Count Dict: {'avocado': 628, 'low fat yogurt': 1421, 'green tea': 2507,
    'honey': 880, 'mineral water': 4700, 'spaghetti': 5072, 'salmon':
    803, 'olive oil': 2045, 'burgers': 1624, 'meatballs': 394, 'eggs':
    3388, 'turkey': 1173, 'milk': 2432, 'french fries': 3202, 'chicken':
    1313, 'frozen vegetables': 2413, 'cooking oil': 1633, 'cookies':
    1508, 'shrimp': 1354, 'pasta': 577, 'chocolate': 3097, 'ground beef':
     2175, 'energy bar': 505, 'escalope': 1479, 'mushroom cream sauce':
    651, 'soup': 958, 'hot dogs': 614, 'brownies': 632, 'muffins': 451, '
    fresh tuna': 420, 'energy drink': 498, 'herb & pepper': 918, '
    champagne': 870, 'ham': 504, 'pancakes': 1754, 'yogurt cake': 1}
Grouping Dict: {614: ['hot dogs'], 628: ['avocado'], 505: ['energy bar
    '], 394: ['meatballs'], 504: ['ham'], 880: ['honey'], 651: ['mushroom
     cream sauce'], 451: ['muffins'], 1633: ['cooking oil'], 4700: ['
    mineral water'], 420: ['fresh tuna'], 1508: ['cookies'], 498: ['
    energy drink'], 870: ['champagne'], 2175: ['ground beef'], 2045: ['
    olive oil'], 958: ['soup'], 632: ['brownies'], 5072: ['spaghetti'],
    918: ['herb & pepper'], 3097: ['chocolate'], 803: ['salmon'], 1173:
    ['turkey'], 1354: ['shrimp'], 577: ['pasta'], 2432: ['milk'], 1754:
    ['pancakes'], 1624: ['burgers'], 2413: ['frozen vegetables'], 3202:
    ['french fries'], 1313: ['chicken'], 1479: ['escalope'], 2507: ['
    green tea'], 3388: ['eggs'], 1: ['yogurt cake'], 1421: ['low fat
    yogurt']}

hit rate: 19865
miss rate: 9498
```

3.6 Improving LFU cache scheme results

To improve the result of the LFU cache scheme we are going to use the results of the apriori algorithm and try to minimize the number of miss which also the number of times the cache is getting modified, and to do that we are going to do the following steps, before adding any item to the cache using **Put** we check if the item is a key in the dictionary containing the mining itemsets we got from the apriori algorithm if it is, than we add both the key and value to the cache. The idea is presented in the following code:

```
  hit=0
miss=0
for i in range(iteration):
  if store_data[i] in lfuCache.cache:
    if store_data[i] in d:
      lfuCache.put2(d[store_data[i]],d[store_data[i]])
      hit=hit+1
    else:
      lfuCache.put(store_data[i],store_data[i])
      hit=hit+1
  else:
    if store_data[i] in d:
      lfuCache.put(store_data[i],store_data[i])
      lfuCache.put2(d[store_data[i]],d[store_data[i]])
      miss=miss+1
    else:
      lfuCache.put(store_data[i],store_data[i])
      miss=miss+1

print("hit rate:",hit)
print("miss rate:",miss)
```

Listing 3.9: Improving LFU cache scheme results

3.7 Discussing the results

In this implementation we used cache sizes that varies from 4% – 20% of the Σ alphabet size, we adopted this sizing from [14] simulation setup

3.7.1 LFU size10 test results:

The 10 size cache represent 8.33% of the number of different item on the dataset, which means that after running the modified LFU using itemsets algorithm most of that space will be taken by the item set extracted using the apriori algorithm, the itemsets from the apriori algorithm don't have the same frequency because some of them are more popular than the other which means they have a higher chance to stay in cache. The result in table 3.2 are represented in chart3.4.

Table 3.2: Size 10 test table

size	10			
	lfu		lfu modified	
iteration	hit	miss	hit	miss
1000	166	834	232	768
3100	554	2546	732	2368
7200	1196	6004	1509	5691
58678	9934	48744	12239	46439
117357	19873	97484	24501	24501

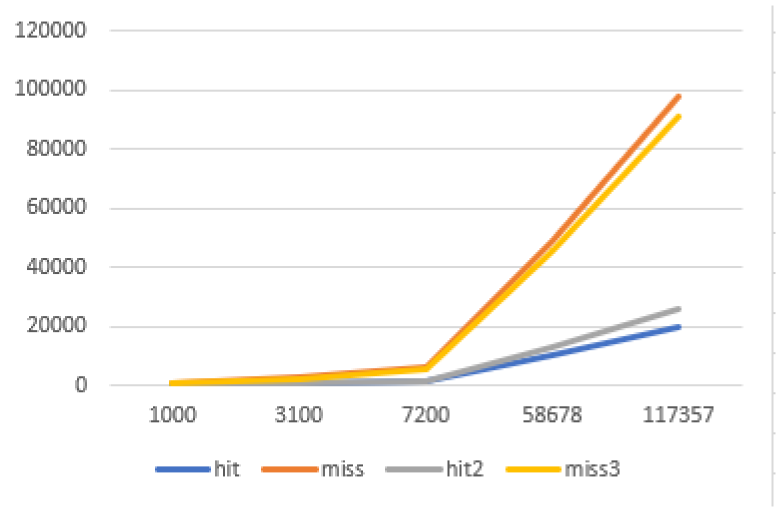


Figure 3.4: Chart representing of the size 10 table

3.7.2 LFU size 18 test results:

The 18 size cache represent 15% of the number of different item on the dataset, since the size of the cache increased the amount of popular itemsets inside the cache increased. The result in table 3.3 are represented in chart3.5.

Table 3.3: Size 18 test table

size	18			
	lfu		lfu modified	
iteration	hit	miss	hit	miss
1000	303	697	385	615
3100	994	2106	1243	1857
7200	2250	4950	2723	4477
58678	18380	40298	22024	36654
117357	36770	80587	44089	73268

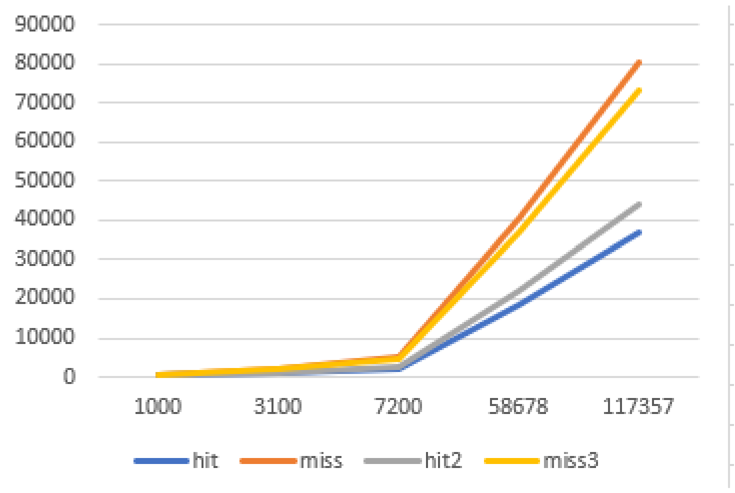


Figure 3.5: Chart representing of the size 18 table

3.7.3 LFU size 24 test results

The 24 size cache represent 20% of the number of different item on the dataset, since the size of the cache increased the amount of popular itemsets inside the cache increased. The result in table 3.4 are represented in chart3.6.

Table 3.4: Size 24 test table

size	24			
	lfu		lfu modified	
iteration	hit	miss	hit	miss
1000	385	615	423	577
3100	1265	1835	1389	1711
7200	2911	4289	3149	4051
58678	23458	35220	25936	32742
117357	46927	70430	51916	65441

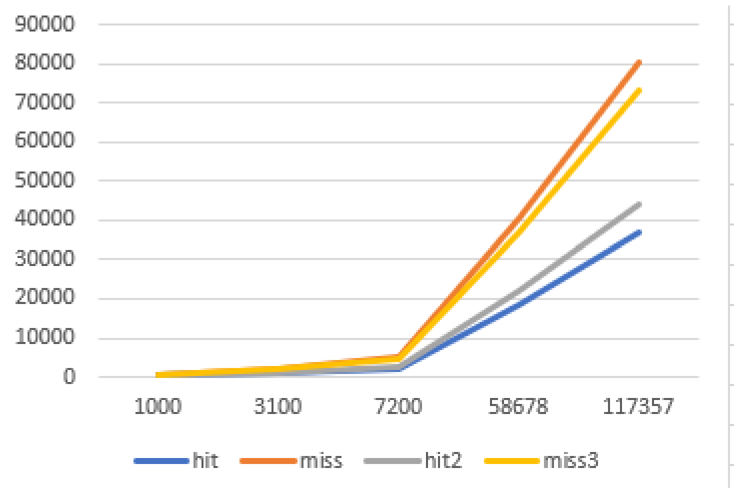


Figure 3.6: Chart representing of the size 18 table

3.7.4 Discussion

In this implementation we used itemsets of two due to the lack of time and resource to improve our implementation, and by doing so we neglected other possible itemsets. For example let $fp \subseteq T$ and $i \subseteq fp$. Lets say fp is a frequent pattern of three items: fp=$\{a_1, a_2, a_3\}$ from fp we get the following subsets:

- $\mathrm{i}_1 = \{\mathrm{a}_1, a_2\}$
- $\mathrm{i}_2 = \{\mathrm{a}_1, a_3\}$
- $\mathrm{i}_3 = \{\mathrm{a}_2, a_3\}$

In this implementation instead of using the three subsets we only used two which affected the results of our implementation. so instead of using items sets of two we could use the whole frequent pattern to get better results.

3.7.5 Conclusion

As a conclusion to this chapter we would like to mention that the itemsets based LFU depends on the number of itemsets and size of each node, so that it can improve its result highly and make it more sufficient.
The results we got from this implementation are about 4% to 11% increase in the hit rate and decrease of the miss rate.
By using frequent patterns instead of itemsets of two items we suppose it can increase the the results to 20% or more depending on the size and iterations The items contained in the itemsets are the ones with highest frequency.
the best results comes from when the cache size is equal or higher to the number of itemsets.

Conclusion

We have presented in this work the basics of frequent pattern mining and used it in cache management. we have also introduced the apriori algorithm which we used to extract frequent patterns from a dataset, we also explained some basic cache placement and replacement schemes.

To save computational time and to reduce the miss rate and raise hit rate the frequent patterns were used to get the data that the user needs before he request it.

After spending much time looking for a dataset to study we tried generating one using the random function, the results were not promising because the item sets where randomly created so they don't describe the users habits, we used the retail dataset to generate some itemsets and tested them on the same dataset and got some good results even though the dataset size was small and the amount of generated itemsets was not that much.

The learning dataset is also a critical element cause it represent the habits of the users we are going to study to extract their frequent patter, so that we could improve their future requests.

In order to achieve this work, we spent a lot of time reading and studying the documents, doing test and trying theories. This work allowed us to put our knowledge of Data mining into practice and to acquire other knowledge, and the time spent reading articles served as a good introduction to research.

As perspectives I am planning to study my experiments on other datasets with bigger size and prove that this results are not random.

Bibliography

[1] Sheneela Naz, Rao Naveed Bin Rais, and Amir Qayyum. A resource efficient multi-dimensional cache management strategy in content centric networks. *Journal of Computational and Theoretical Nanoscience*, 15:1137–1152, 04 2018.

[2] Anxiao Jiang and Jehoshua Bruck. Optimal content placement for en-route web caching. *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.*, pages 9–16, 2003.

[3] Chih Yen Chang and Ming Sang Chang. A hybrid coordination approach of in-network caching for named data networking. *International Journal of Future Generation Communication and Networking*, 9(4):285–300, 2016.

[4] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60. ACM, 2012.

[5] bigdatanerd. Introduction to data mining – Types of data mining techniques, June 2011. Library Catalog: bigdatanerd.wordpress.com.

[6] Adelaja Oluwaseun and Mani Chaubey. Data mining classification techniques on the analysis of student's performance. 7:17, 04 2019.

[7] Piyush Tyagi. Clustering, December 2018. Library Catalog: medium.com.

[8] Yusung Kim and Ikjun Yeom. Performance analysis of in-network caching for content-centric networking. *Computer Networks*, 57(13):2465 – 2482, 2013.

[9] G. Prabaharan and S. Jayashri. Mobile cluster head selection using soft computing technique in wireless sensor network. *Soft Computing*, 23(18):8525–8538, September 2019.

[10] Rick-Anderson. Distributed caching in ASP.NET Core. Library Catalog: docs.microsoft.com.

[11] Hierarchical Caching — Apache Traffic Server 10.0.0 documentation.

[12] Xueyan Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6):595–607, June 2002.

[13] Named Data Networking (NDN) - A Future Internet Architecture. Library Catalog: named-data.net.

[14] Faiza Qazi, Osman Khalid, Rao Naveed Bin Rais, Imran Ali Khan, et al. Optimal content caching in content-centric networks. *Wireless Communications and Mobile Computing*, 2019, 2019.

[15] Zhongxing Ming, Mingwei Xu, and Dan Wang. Age-based cooperative caching in information-centric networks. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 268–273. IEEE, 2012.

[16] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[17] Thomas H. Davenport. Machine learning. https://www.sas.com/en_us/insights/analytics/machine-learning.html, 2019.

[18] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 01 2012.

[19] Swati Gupta. A Regression Modeling Technique on Data Mining. *International Journal of Computer Applications*, 116(9):27–29, April 2015.

[20] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[21] Hannu Toivonen. *Apriori Algorithm*, pages 39–40. Springer US, Boston, MA, 2010.

[22] What is Python? Executive Summary. Library Catalog: www.python.org.

[23] Z. Li and G. Simon. Time-shifted tv in content centric networks: The case for cooperative in-network caching. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6, June 2011.

[24] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The lcd interconnection of lru caches and its analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[25] Michael Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: using remote client memory to improve file system performance. In *OSDI '94*, 1994.

[26] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing global memory management in a workstation cluster. *SIGOPS Oper. Syst. Rev.*, 29(5):201–212, December 1995.

[27] Tutorials Point. Artificial intelligence neural networks. https : / / www.tutorialspoint.com / artificial_intelligence / artificial_intelligence_neural_networks.htm, May 2019.

[28] Aish Warya. Introduction to recurrent neural network. https : / / www.geeksforgeeks.org/introduction-to-recurrent-neural-network/.