Parallel Itemsets Mining in Massively Distributed Environments

Mehdi Zitouni

Thesis

defended to obtain the diploma of
Ph.D. in Computer Sciences

Elaborated by
Mehdi Zitouni
(Master's degree in computer sciences, Faculty of Science of Tunis)

Parallel Itemsets Mining in Massively Distributed Environments

Thesis Committee

M. Faouzi Moussa	Professeur, Faculté des Sciences de Tunis	President
M. Arnaud Giacometti	Professeur, Université François Rabelais de Tours	Rapporteur
M. Mounir Ben Ayed	Maitre de conference, Faculté des Sciences de Sfax	Rapporteur
M. Reza Akbarinia	Directeur de recherches, INRIA Montpellier	Co-directeur
M. Florent Masseglia	Directeur de recherches, INRIA Montpellier	Co-directeur
M. Sadok Ben Yahia	Professeur, Faculté des Sciences de Tunis	Directeur

5th December 2018

To my parents ...
To my grandmother ...
To my future wife ...
To my family ...
To my friends ...

Acknowledgements

As a 25-year old student I decided that my dissertation would be finished by the age of 28. Here I am, at 31, and still finishing my dissertation. Things have not gone as planned. However, life never goes according to plans, otherwise it would be quite boring. Now that this work is finally coming to an end, I have many people to thank.

I firstly would like to thank my co-advisor Dr ***Florent Masseglia*** *for his brilliant work in guiding this thesis. Thank you Florent. Your professional instruction has been excellent. Your supportive encouragement when I have been struggling with my dissertation has been very important. Thank you for your patience and support, both in my career and in my personal life. I highly respect your professional advices, but most of all I respect your friendship during times when I needed help, support, more encouragement, to finally finish my work. I would like to thank also my second co-advisor, Dr* ***Reza Akbarinia****, without whom, his advises and his ingenious guiding, I would be still searching and searching for the solution. Thank you Reza for your kindness and professional manners throughout my thesis. I gratefully thank you for your presence and support, always with your smile, in all the moments of joy and stress during my presence in the Zenith Team. I would like also to express my sincere gratitude to Pr.* ***Sadok Ben Yahia****, who has been supervising my work since 2011 (7 years!) and with whom I did my very first research internships in my Master's degree. Sir, I thank you for the continuous support of my study and related research, and for your patience, motivation and immense knowledge. The confidence you has always shown me, have been a great contribution throughout my work. Sir, I will never forget that it was thanks to you that I am doing research today.*

My sincere thanks go also to my research team, the Zenith Team of INRIA, who provided me an opportunity to join their team as an intern, and who gave me the

access to the laboratory and research facilities. Without their precious support, it would not be possible to conduct this research.

A special thanks to my family, my parents Semia and Ali, my brothers Hamza and Fadhel and my sweet and most lovely little sister Nesrine. Words cannot express how grateful I am. Your supports, encouragements, prayers and reassurements for me have been what have sustained me so far.

A grateful thanks to my great family. To my sweet grandmother Om Saad, for her kindness and tenderness. To my uncle Nouri for his help, his strengthen support and encouragements, you are the best Nouri. To my aunt Wafa, a second mother and a great friend. To my passed away uncle Ali, who was often in my thoughts on this journey, a second father and a generous friend, you are missed. To my future mother in law Monjeya and my sweet aunt Youva. To my many cousins and the rest of my big family, you should know that your supports and encouragements have been worth more than I can express on paper.

I am also indebted to all my friends for various reasons related to finally finishing my thesis, some of whom deserve special mention. A special gratitude to my big bro Rafaa, thanks to who I have been always well advised and put in the right direction. To my bro Anis, for his encouragements and bad jokes that even when I am depressed, they make me laugh. Thank you Miaou, Bolbol, Djo, T-bag, Bassou and Hamma for your sacrifices and your comfort. You have been always there for me and for my goodness. Huge thanks to my dearest friend Fatma for her constant support, her effort in order to promote this lecture and lead it to take place. Thank you Teresa, Nathalie, Monique, Hallouma, Imen, Amina, Malek and all who has participated from far or near to realize this work.

Finally, my biggest gratitude goes to my present work team at Pikcio in Montpellier, France. Thank you Jorick, Fabien, Benjamin, Delphine, Mostafa, Alizon, Marie, Patrick, Raphaël, Paul, Antoine, Magali, Véro and all the rest of the team around the world. I thank you very much for your support and your warm welcoming me into your professional, kind and lovely team.

Mehdi ZITOUNI

Résumé

Les progrès dans les technologies matérielles et logicielles et leurs évolutions ont offert une grande flexibilité pour produire et stocker des quantités très grandes de données qui ne cesse de croître. À tel point qu'on parle aujourd'hui de "Big Data".

Les méthodes d'analyse de données ont toujours été confrontées à des quantités qui mettaient en difficulté les capacités de traitement, ou qui les dépassaient. Pour franchir les verrous technologiques associés à ces questions d'analyse, la communauté peut se tourner vers les techniques de calcul distribué. En particulier, l'extraction de motifs, qui est un des problèmes les plus abordés en fouille de données, présente encore souvent de grandes difficultés dans le contexte de la distribution massive et du parallélisme. Dans cette thèse, nous abordons deux sujets majeurs liés à l'extraction de motifs, les motifs fermés fréquents, et les motifs informatifs (*i.e.*, de forte entropie).

L'analyse des données en général, et les primitives d'exploration de données en particulier, sont une source majeure de goulots d'étranglement dans le fonctionnement des systèmes d'information. Ceci est principalement dû à leur grande complexité et à leur utilisation intensive des opérations d'accès au disque, en particulier dans les environnements massivement distribués. En outre, une application majeure issue des analyses de données consiste à découvrir des informations clés sur les traces du système d'information afin d'améliorer leur ingénierie. L'extraction des itemsets fermés fréquents ($\mathcal{CFI}$) est l'une de ces techniques d'exploration de données, associée à de grands défis.

Le problème de l'extraction d'itemset fréquents fermés dans les données massives s'est imposé depuis des décennies. Dans le cadre des travaux entrepris dans cette thèse, nous définissons des techniques d'analyse de données spécifiques, en adoptant une approche basée sur la codification en nombres premiers dans les

datasets, dans des environnements massivement distribués afin d'améliorer les performances du processus d'extraction des itemsets fréquents fermés en parallèle ($\mathcal{CFI}$). Nous introduisons DCIM (Distributed Closed Itemsets Mining), un algorithme parallèle pour extraire les $\mathcal{CFI}$s d'une énorme quantité de données. DCIM permet de découvrir l'ensembles des itemset fréquents avec une meilleure efficacité et une compacité des résultats. Une caractéristique clé de DCIM est la combinaison profonde des propriétés d'exploration de données avec les principes de la distribution massive de données. Nous avons réalisé des expériences exhaustives sur des jeux de données du monde réel, des datasets contenant jusqu'à 53 millions de documents, pour illustrer l'efficacité de DCIM.

Dans un deuxième temps, nous nous intéressons au problème de la découverte des motifs informatifs maximales de taille k (*miki* ou "maximally informative k-itemsets) à partir d'un flux de données. Nous proposons PENTROS (Parallel Entropy computing over streams), un algorithme pour leur extraction en environnement dynamique et distribué. PENTROS rend le processus d'extraction de *miki* dans des grandes quantités entrantes de données simple et efficace. Avec PENTROS, nous proposons un ensemble de techniques d'optimisation pour calculer l'entropie conjointe des motifs de différentes tailles. Ceci permet de réduire le taux de latence du processus d'extraction dans le streaming de manière significative. PENTROS a été évalué en simulant des streaming à partir des données massives du monde réel. Les résultats de nos expérimentations confirment l'efficacité de notre approche par le passage à l'échelle de notre approche sur des motifs de grande taille, à partir de très grandes volumes de données entrantes et sortantes.

Par ailleurs, la classification est l'une des briques les plus importantes de la fouille de données et de la recherche d'information. Le problème de classification a été largement étudié dans des environnements centralisés. Cependant, dans les environnements massivement distribués, les algorithmes de classification necessitent une profonde exploitation pour améliorer leur temps d'exécution et leur précision. À cette fin, notre motivation derrière l'extraction des motifs informatifs repose sur le fait qu'ils peuvent être utilisés pour paramétrer efficacement les algorithmes de classification et gagner en terme précision. Ainsi, un déploiement des patterns informatifs comme modèle de *feature selection* pour les algorithmes de classification supervisée sera nécessaire pour esquisser l'amélioration en taux de précision. Ainsi, dans la troisième contributionn de cette thèse, nous abordons le problème de la classification parallèle dans des environnements hautement distribués. Nous proposons

EEC (Ensemble of Ensembles of Classifier) pour rendre la tâche de la classification simple et plus efficace. L'algorithme est composé de deux Jobs Spark. Combinant plusieurs classificateurs, EEC exploite profondément le parallélisme sous le framwork Spark pour non seulement réduire le temps d'exécution mais aussi améliorer de manière significative la précision de la classification en effectuant deux étapes de prise de décision. Nous montrons que la précision de la classification de EEC a été améliorée en utilisant des modèles informatifs et que l'erreur de classification peut être limitée à une petite valeur dans un environnement parallèle. EEC a été évalué en utilisant les jeux de données "English Wikipedia articles" et "clue Web". Nos résultats expérimentaux montrent que EEC est significativement plus efficace et précis que les approches pionnières de la littérature.

Titre en français

L'Extraction des motifs dans des Environnements Massivement Distribués

Mots-clés

- Extraction de motifs
- Données massives
- Analyse de concepts formels
- Données distribuées
- Classification supervisée

Abstract

For the last few decades, the volume of data has been increasingly growing. The rapid advances that have been made in computer storage have offered a great flexibility in storing very large amounts of data in which, the mining process is still facing numerous challenges for discovering knowledge and hidden correlations.

Data analytics in general, and data mining primitives in particular, are a major source of bottlenecks in the operation of information systems manipulating the huge amount of generated data. This is mainly due to their high complexity and intensive call to IO operations, particularly in massively distributed environments. Moreover, an important application of data analytics is to discover key insights from the running traces of the information system in order to improve its engineering. Mining Closed Frequent Itemsets ($\mathcal{CFI}$) is one of these data mining techniques, associated with great challenges.

Despite the answers that frequent itemset mining methods can provide about data, some hidden relationships cannot be easily driven and detected inside data. This is specifically the case when data are very large and massively distributed. To this end, a careful analysis of the informativeness of the itemsets would give more explanation about the existing correlations and relationships inside data. However, digging through very large amounts of data to determine a set of maximally informative itemsets (of a given size k) presents a major challenge in data mining. This is particularly the case as far as the size k of the informative itemsets to be discovered is very high.

Moreover, classification is one of the building bricks in data mining and information retrieval. The problem has been widely studied in centralized environments. However, in massively distributed environments, parallel classification algorithms have not gained much in terms of accuracy. To this end, our motivation behind the mining of informative patterns lies in the fact that they can be used effectively

as a feature selection application in classification algorithms. Indeed, from the literature, the informative patterns can show a major improvement in classifying documents and transactions. Thus, a deep modeled deployment of informative patterns as features for supervised classification use cases will be needed to sketch the major improvement in accuracy rates in those use cases.

In the beginning of this thesis, we tackle the problem of $\mathcal{CFI}$ mining in big datasets. We adopt a prime-number-based approach to improve the performance of a parallel $\mathcal{CFI}$ mining process. We introduce Distributed-Closed-Itemset-Mining (Dcim), a parallel algorithm for mining $\mathcal{CFI}$s from large amounts of data. Dcim allows discovering itemsets with better efficiency and result compactness. A key feature of Dcim is the combination of data mining properties with the principles of massive data distribution. Exhaustive experiments are carried out over real world datasets to illustrate the efficiency of Dcim for large real world datasets with up to 53 million documents.

The second problem we address in this thesis is the discovery of maximally informative k-itemsets (*miki*) from a huge incoming/outgoing data over a stream based on joint entropy. We propose Parallel entropy computing over Streams (PentroS) a highly scalable, parallel *miki* mining algorithm that renders the mining process of the large throughput of data succinct and effective over a data streaming process. Its mining process is made up of only two efficient parallel jobs. With PentroS, we provide a set of significant optimizations for computing the joint entropy of the *miki* having different sizes, which drastically reduces the latancy rate of the mining process. PentroS is extensively evaluated using a massive real-world data stream. Our experimental results confirm the effectiveness of our proposal by the significant scale-up obtained with lengthy itemsets and over very large throughputs of data.

Finally, we address the problem of parallel classification in highly distributed environments. We propose Ensemble of Ensembles of Classifiers (Eec), a parallel, scalable and highly accurate classifier algorithm. Eec renders a classification task simple, yet very efficient. Its working process is composed of two simple and compact jobs. Calling to more than one classifier, Eec cleverly exploits the parallelism setting not only to reduce the execution time but also to significantly improve the classification accuracy by performing two level decision making steps. We show that the Eec classification accuracy is improved by using informative patterns and that the classification error can be bounded to a small value. Eec is extensively

evaluated using various real-world, large data sets. Our experimental results suggest that Eec is significantly more efficient and more accurate than alternative approaches.

Title in English

Parallel Itemsets Mining in Massively Distributed Environments

Keywords

- Pattern mining
- Massive data
- Formal concepts
- Data distribution
- Supervised classification

Laboratory

LIPAH - Laboratoire d'informatique et de programmation algorithmique et heuristique.

Adress

Campus Universitaire de Tunis ElManar
Faculté des sciences de Tunis
Département des sciences de l'informatique
2092 ElManar, Tunis

Research Team

Zenith Team, INRIA

Adress

Université Montpellier
Bâtiment 5
CC 05 018
Campus St Priest - 860 rue St Priest
34095 Montpellier cedex 5

Résumé étendu

Introduction

Dans le monde professionnel, des problèmes très divers s'imposent dans la gestion des données étant d'une grande quantité et très diversifié, allant de la gestion de la relation client, à la maintenance préventive, en passant par la détection de fraudes ou encore l'optimisation de sites web. Ici, le terme Extraction des connaissance s'impose.

Étant un terme historique (2238 av. J.-C.), l'exploration des données a bien convergé selon les générations par lesquelles elle passait. Ce n'est qu'en 1662 que John Graunt [1] publiait son livre analysant la mortalité à Londres et essayait de prévoir les apparitions de la peste bubonique. En 1763, Thomas Bayes [2] montre qu'on peut fixer, non seulement, des probabilités à partir des observations conclues d'une expérience, mais aussi les paramètres relatifs à ces probabilités, etc. Arrivant aux années 80 ou Rakesh Agrawal [3] employa le terme entamant une recherche sur des bases de la taille de 1Mb.

De nos jours, ce processus, se manifestant avec une très remarquable importance, se définit comme étant un acteur ayant pour objet d'illustrer un savoir, une connaissance, à partir des bases de données extrêmement massives et labyrinthées. Il fait suite, dans l'escalade de l'exploitation des données de l'entreprise, à l'informatique décisionnelle. Et pour traiter des données aussi volumineuses, une solution consiste à les distribuer sur plusieurs machines et les traiter en parallèle. En fouille de données, cette solution exige une révision profonde des différents algorithmes, pour qu'ils deviennent capables de traiter les données massives en parallèle.

La fouille de données est le domaine de recherche au sein duquel coopèrent statisticiens, spécialistes en bases de données et en intelligence artificielle, ou encore chercheurs en conception d'interfaces homme-machine. Les informations extraites, suite à l'application d'un processus de fouille de données, peuvent prendre plusieurs formes, telles que les règles associatives, les arbres de décisions, les réseaux de neurones, etc. En s'intéressant à la technique d'extraction des règles associatives, elle se décompose en deux problèmes majeurs à résoudre: i) l'extraction des motifs (ou itemsets) fréquents, ayant une fréquence de co-occurrence supérieur ou égale à un seuil fixé par l'utilisateur; ii) la génération de toutes les règles d'association valides possible à partir des motifs extraits respectant un seuil d'intérêt fixé par l'utilisateur. C'est ainsi que la fréquence de co-occurrence des variables d'un motif présente une mesure d'information, qui permet de déterminer l'utilité d'un tel motif en se basant sur sa fréquence d'apparition dans la base des données. L'extraction des motifs fréquents présente de nombreux domaines d'applications. Par exemple, en fouille de texte [4], une technique d'extraction des motifs fréquent peut être utilisée pour déterminer les mots qui se répètent fréquemment dans une grande base des données. En commerce électronique, une telle technique peut être utilisée pour recommander des produits comme des livres, des vêtements, etc. À cet effet, les auteurs de [5] ont proposé un algorithme de type *tester et générer*, nommé Apriori conçu pour extraire les motifs fréquents à partir des bases de données transactionnelles. À ce stade, nous remarquons le nombre très élevé des motifs fréquents, ce qui a l'inconvénient de perturber leur exploitation et de rendre leur interprétation, par des experts humains, quasi-impossible. En conséquence, la réduction de ce grand nombre des motifs fréquents est devenu d'une importance primordiale pour une meilleur exploitation des informations extraites à partir des bases de données massives.

Plusieurs travaux ont été conçus pour définir des ensembles de motifs qui soient les plus compacts possibles. Un tel ensemble est appelé *représentation concise exacte* des motifs fréquents. Dans la littérature, les représentations concises exactes ont été définies a base de plusieurs et différentes astuces, la première était l'exploration des motifs fermés fréquents. Cette approche a été proposée afin de palier plusieurs inconvénients dont la plus remarquable est la réduction du nombre très élevé des motifs fréquents à explorer. Cette approche est issue des fondements mathématiques de l'analyse formelle de concepts introduite par [6], et elle a permis une réduction très notable dans le coût d'extraction des motifs fréquents. Plusieurs algorithmes pour fouiller les itemsets fermes fréquents ont été proposés dans la littérature. Dans [7], l'algorithme Apriori-Close adopte le meme principe de fouille que Apriori. En effet, pour aboutir à un résultats final composé d'une liste complète des itemsets fermes frequents, Apriori-Close traite les itemsets de l'espace de recherche un par un commençant de la taille 1 jusqu'à $\mid \mathcal{I} \mid$. C'est ainsi que nous pouvons remarquer qu'une implémentation basique de A-Close dans un environnement parallel pourra posé de grands problème, à savoir le nombre de jobs très élevé pour traiter les itemsets de l'espace de recherche avec un coût de communication qui s'est avéré très élevé vu les opérations d'élagage que l'algorithme A-Close impose dans l'espoir de réduire

l'ensemble des itemsets candidats potentiellement fermés et fréquents. FP-Close est introduit dans [8]. De même principe que FP-Growth, FP-Close utilise l'opérateur de fermeture pour déduire l'ensemble des fermés à partir des FP-Trees construites, etc.

Toutefois, dans certaines applications, analyser les données en se basant sur la fréquence de co-occurrences des variables comme une mesure d'information n'aboutit pas forcément à des résultats pertinents. La fréquence de co-occurrences des variables n'aide pas à capturer tous les motifs intéressants et informatifs dans la base des données. En particulier, c'est le cas quand les données sont creuses ou qu'elles répondent à une distribution large. Dans de tels cas, d'autres mesures d'information des motifs peuvent être prises en compte. Une mesure d'information intéressante pour les motifs est l'entropie (plus précisément l'entropie pour une variable et l'entropie conjointe pour un motif, qui est un ensemble de variables). Le motif de taille k qui a une valeur d'entropie maximale parmi les autres motifs (de même taille k), serait considéré comme un motif discriminant de taille k (*miki*, ou maximally informative k-itemset). Les items qui composent un tel motif discriminant sont faiblement corrélés entre eux, mais si on les considère tous ensemble, ces items divisent les enregistrements de la base de donnéesvde manière très optimale. Les motifs discriminants ont des applications dans plusieurs domaines différents. Par exemple, en classification, ils peuvent être utilisés pour déterminer les attributs indépendants les plus pertinents dans la base d'apprentissage.

Avec la disponibilité des modèles de programmation performants comme MapReduce [9] et Spark [10], le traitement des données de masses devient une tache facile à accomplir. Cependant, la plupart des algorithmes parallèles de la fouille des données souffrent encore de plusieurs problèmes. En particulier, les algorithmes parallèles d'extraction des motifs fréquents souffrent des mêmes limitations que leurs implémentation séquentielle. Ces différentes limitations sont fortement liées à la logique et aux principes de fonctionnement de chaque algorithme. Par exemple, l'implémentation centralisée (i.e., séquentielle) de l'algorithme Apriori [5] demande plusieurs accès au disque. Une version parallèle de cet algorithme, avec une implémentation directe qui considère les jobs MapReduce comme une interface remplaçant les accès au disque, présenterait les mêmes inconvénients (i.e., la multiplication des jobs pour valider les différentes générations de motifs serait un goulot d'étranglement). Enfin, bien que l'algorithme FP-Growth [11], ait été considéré comme l'algorithme le plus efficace pour l'extraction des motifs fréquents, avec un très faible support minimum et très grand volume des données, sa version parallèle Pfp-Growth [12] n'est pas capable de passer à l'échelle à cause de sa consommation en mémoire d'autant qu'en augmentant la taille des motifs k à fouiller.

De la même manière, les algorithmes d'extraction des motifs discriminants n'échappent pas à cette difficulté d'adaptation du centralisé vers le parallèle. L'extraction des *miki* en parallèle n'est pas une tâche facile. Le calcul parallèle de l'entropie est coûteux en raison du grand nombre d'accès au disque dont il a besoin. Par exemple, considérons une version parallèle de l'algorithme ForwardSelection [13], pour déterminer les *miki*, ForwardSelection aurait besoin de k jobs en parallèle.

En plus des problèmes de traitement liés à la découverte de motifs, dans des environnements massivement distribués, la quantité de données transférées peut affecter la performance globale. La conception d'algorithmes d'extraction des motifs fréquents ou discriminants en parallèle doit alors considérer cette question, pour optimiser le coût de communication des données dans les environnements distribués.

Dans la suite, nous présentons des exemples de problèmes qu'un algorithme parallèle de fouille des données pourrait avoir quand il traite des grandes quantités des données. En particulier, nous nous concentrons sur les algorithmes d'extraction des motifs fermés fréquents et les algorithmes d'extraction des *miki*s.

Example 1 *Considérons un support minimum très petit, supposons que nous voulons déterminer les itemsets fermés fréquents dans une base de données $\mathcal{D}$ très large en utilisant une version parallèle de l'algorithme* Apriori-Close. *Le nombre de jobs MapReduce serait proportionnel à la taille du motif candidat le plus long dans la base de données $\mathcal{D}$. En général, dans un environnement massivement distribué, cette approche qui consiste à un scan multiple de $\mathcal{D}$ aboutit à une mauvaise performance. En particulier, le nombre des données (les motifs candidats) transférées entre les mappers et les reducers serait très grand.*

Maintenant, considérons une version parallèle de l'algorithme FP-Close *pour extraire les motifs fermés fréquents dans la base de données $\mathcal{D}$. Avec le même très petit minimum support et une recherche exhaustive de motifs fermés fréquents (le paramètre k prend une valeur infinie), l'algorithme serait souffert de plusieurs limitations. La taille de l'arbre FP-Tree pourrait être très grande et donc dépasse la capacité de la mémoire. Si n'est pas le cas, la quantité des données transférées serait très grande ce qui affecte la performance globale du processus d'extraction des motifs fermés fréquents.*

Example 2 *Dans cet exemple, supposons que nous voulons déterminer les motifs informatifs maximaux de taille k. Considérons une version parallèle de l'algorithme* ForwardSelection. *Ça depend au taille k des motifs à découvrir, l'algorithme s'exécute en k jobs de MapReduce. En effet, les performances de l'algorithme seraient très pauvres. En plus, le nombre des candidats des motifs informatifs maximaux serait très grand. Donc, l'extraction parallèle des motifs informatifs maximaux de taille k tombe dans les mêmes limitations et restrictions de celles de l'extraction parallèle des motifs fréquents.*

Par ailleurs, le problème de classification en machine learning [14] est l'une des briques de construction de la fouille de données et de la recherche d'information. En effet, la classification est un processus d'apprentissage supervisé qui consiste en l'affectation automatique d'une instance à une catégorie prédéfinie (e.g. Classement d'une seule instance) ou plus (classification multi-étiquettes). Le processus de classification se manifeste en tant qu'un processus d'apprentissage de machine (e.g. construction d'un modèle ou d'un classifieur) capable de prendre des décisions sur la base de son historique. En bref, le problème de la classification peut être défini comme suit: Étant donné un ensemble de données

d'apprentissage avec un nombre fixe d'instances étiquetées (i.e. Chaque instance a déjà été affectée à une catégorie prédéfinie), nous proposons de créer un modèle qui permettra de classer une nouvelle instance dans une catégorie appropriée avec un taux d'erreur de classification très réduit.

De nos jours, nous sommes complètement débordés par les données provenant de différentes sources telles que les réseaux sociaux, les capteurs, etc. Pour traiter ce gros volumes de données, les algorithmes conventionnels de classification ont montré leurs limites. Généralement, les données ne peuvent pas être stockées dans la mémoire, et même si c'est le cas, le processus de classification s'avère coûteux en terme d'entrées/sorties et de temps d'exécution. Par conséquent, les algorithmes de classification ne sont plus en mesure de traiter efficacement (en terme de précision) une grande quantité de données dans des environnements centralisés.

Dans cette thèse, nous abordons le problème de la classification parallèle dans des environnements hautement distribués. Nous proposons EEC (Ensemble of Ensembles of Classifiers), un algorithme de classification parallèle, évolutif et très précis. EEC rend la tâche de classification très simple, voire très efficace. L'algorithme est composé de deux Jobs Spark simples et compactes. Appelant plus d'un classifieur, EEC exploite intelligemment le paramétrage du parallélisme pour non seulement réduire le temps d'exécution mais aussi améliorer de manière significative la précision de la classification en effectuant deux étapes de prise de décision. Nous montrons que la précision de la classification de EEC a été très améliorée en utilisant les motifs informatives et que l'erreur de classification des instances peut être limitée à une petite valeur. EEC a été évalué de manière très approfondie en utilisant des jeux de données très volumineux. Nos résultats expérimentaux montrent que EEC est significativement plus efficace et précis que les approches alternatives de la litterature.

État de l'art

Extraction parallèle des motifs fermés fréquents

Le problème d'extraction des itemsets fréquents a été d'abord introduit dans [5].

Definition 1 *Soit $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ un ensemble qui contient des éléments s'appellent items. Un $Motif$ X est un ensemble des items de $\mathcal{I}$, i.e. $X \subseteq \mathcal{I}$. La taille (size) de X égale à son nombre des items qu'il contient. Une transaction T est un ensemble des items tel que $T \subseteq \mathcal{I}$ and $T \neq \emptyset$. Une transaction T supporte un item $x \in \mathcal{I}$ si $x \in T$. Une transaction T supporte un motif $X \subseteq \mathcal{I}$ si elle supporte tous les item $x \in X$, i.e. $X \subseteq T$. Une base de données (database) $\mathcal{D}$ est un ensemble des transactions. Le support d'un motif X dans la base de données $\mathcal{D}$ est égal au nombre total des transactions $T \in \mathcal{D}$ qui contiennent X. Un motif $X \subseteq \mathcal{I}$ est dit fréquent ($frequent$) dans $\mathcal{D}$ si son support est supérieur ou égal à un seuil de support minimum ($MinSup$). Un motif fréquent maximal*

est un motif fréquent qui n'est pas inclus dans aucun autre motif fréquent.

Tid	Transactions
1	C, D, E
2	B, C, E
3	A, B, C, E
4	B, E
5	A, B, D
6	A, B, C, E
7	B, C, D, E

Figure 1: Base de données $\mathcal{D}$

Une approche naïve pour déterminer tous les motifs fréquents dans une base de données $\mathcal{D}$ consiste simplement à déterminer le support (*support*) de toutes les combinaisons des items dans $\mathcal{D}$. Ensuite, garder seulement les items/motifs qui satisfaitent un seuil de support minimum ($MinSup$). Cependant, cette approche est très coûteuse, car elle impose plusieurs accès à la base des données.

Example 3 *considérons la base de données$\mathcal{D}$ qui contient 7 transactions comme illustré par la Figure 1. Avec un seuil de support minimum égal à 7, il n'y a pas des items fréquents (par conséquent, pas des motifs fréquents). Par contre, avec un seuil de support minimum égal à 5, il y a cinq motifs fréquents:* $\{B, C, E, BE, CE\}$

Dans la littérature, plusieurs algorithmes ont été proposés pour résoudre le problème d'extraction des motifs fréquents [5], [15] , [16], [17], [18], [19], etc. Malgré la différence entre leurs principes et logiques, l'objectif de ces algorithmes est d'extraire des motifs fréquents dans une base de données$\mathcal{D}$ en respectant un seuil de support minimum ($MinSup$).

En revanche, quand la taille de la dataset est très élevée, le nombre des motifs fréquents devient notablement très grand, ce qui perturbera leur exploitation et rendra leur interprétation, par des experts humains, quasi-impossible. D'où la réduction de ce grand nombre des motifs fréquent est devenu d'une importance primordiale pour une meilleure concession des informations extraites à partir des bases. Dans la définition suivante nous définissons l'ensemble des itemsets fermés fréquents.

Definition 2 ***Itemsets fermés*** *Etant donné un opérateur de fermeture de la connexion de Galois qu'on prénommera ϕ, un itemset $l \subseteq \mathcal{I}$ tel que $\phi(l) = l$ est appelé itemset fermé. Un itemset fermé est donc un ensemble maximal d'items communs à un ensemble d'objets [20].*

Example 4 *Considérons la base de données $\mathcal{D}$ illustrée par la figure 1, l'itemset {ABCE} est un itemset fermé puisqu'il est l'ensemble maximal d'items communs aux objets {3,5}.*

L'itemset {BC} n'est pas un itemset fermé car il n'est pas un ensemble maximal d'items communs à certains objets: tous les objets contenant les items B et C, i.e., les objets 2, 3, 6 et 7 contiennent également les items A, E et D.

Definition 3 ***Itemsets fermés fréquents*** *Un itemset fermé l est dit fréquent si son support relatif excède un seuil minimum fixé par l'utilisateur noté MinSupp [20].*

Agrawal *et al.* ont introduit dans [21], les deux propriétés suivantes relatives aux supports des itemsets fréquents:

1. Tous les sous-ensembles d'un itemset fréquent sont fréquents.
2. Tous les sur-ensembles d'un itemset infréquent sont infréquents.

Ces propriétés restent applicables dans le cas des itemsets*fermés* fréquents [20]. Ainsi,

1. Tous les sous-ensembles d'un itemset fermé fréquent sont fréquents.
2. Tous les sur-ensembles d'un itemset fermé infréquent sont infréquents.

Au début de nos travaux de recherche, rares étaient les solutions pour la fouille des itemsets fermés fréquents dans des environnements hautement distribués en utilisant MapReduce pour traiter de très grandes quantités de données. Dans ce qui suit, nous focalisons notre étude sur des algorithmes parallèles et séquentiels (dédiés à la parallélisation) d'extraction des motifs fermés fréquents. En particulier, on limite notre étude aux approches qui seront en relation avec ce travail de thèse.

Parallel FP-Growth: Une version parallèle de l'algorithme FP-Growth est PFP-Growth a été proposée dans [12]. PFP-Growth est considéré parmi les algorithmes parallèles d'extraction des motifs fréquents les plus performants. Dans son premier job, PFP-Growth détermine les items qui sont fréquents dans la base des données. Dans le deuxième job, l'algorithme construit un arbre FP-tree pour être exploré après dans le reducer. Le processus de fouille de PFP-Growth se continue dans la mémoire ce qui explique sa haute performance.

Malgré sa performance, avec un seuil de support minimum très petit et grand volume des données, PFP-Growth ne passe pas à l'échelle. Ce comportement de PFP-Growth sera mieux illustré par nos expérimentations dans le chapitre 4. La raison derrière cette limitation de PFP-Growth est hautement liée à l'espace mémoire nécessaire pour traiter un grand nombre d'itemsets de taille k très haute.

P-Closet: L'algorithme Closet utilise une structure de données avancée, basée sur la notion de *trie*, appelée *arbre FP-Tree* [11]. La particularité de cette structure réside dans le fait que plusieurs transactions partagent un même chemin, de longueur n dans l'arbre *FP-Tree*, s'ils ont les n premiers items en commun. L'algorithme Closet effectue le processus d'extraction des itemsets fermés fréquents en deux étapes successives [22].

Une première étape qui se focalise sur la construction de l'arbre *FP-Tree* où les items des transactions sont ordonnés dans un ordre décroissant de support après avoir élagué les items infréquents. Pour chaque transaction du contexte, les items sont traités et une branche est créée suivant le besoin. Dans chaque nœud de la structure *FP-Tree*, il y a un compteur qui garde la trace du nombre de transactions partageant ce nœud. La deuxième étape est dédiée à l'exploration de l'arbre *FP-Tree*. Ainsi, il commence par considérer les 1-itemsets fréquents, triés par ordre croissant de leurs supports respectifs, et examine seulement leurs *sous-contextes conditionnels* (ou *FP-Tree conditionnels*) [22]. Un sous-contexte conditionnel ne contient que les items qui co-occurrent avec le 1-itemset en question. Le *FP-Tree* conditionnel associé est construit et le processus se poursuit d'une manière *récursive*. Par conséquent, nous proposons la version de P-Closet, un algorithme parallèle conçu et mis en place en utilisant le modèle MapReduce. En effet, après une phase de pré-traitement des transactions de la base, chaque mapper dans notre architecture prendra un sous context conditionnel et commencera le processus d'extraction des itemsets fermés fréquents indépendamment des autres mappers. Ceci dit, avant la distribution des données nous avons mis en place une approche de classification des transactions nous permettant d'établir une distribution en *split* des groupes de transactions indépendants des autres groupes.

Extraction parallèle des motifs informatifs

A	B	C	D
1	0	1	0
1	0	1	0
1	0	0	0
1	1	0	0
1	1	1	0
0	0	1	0
0	0	0	0
0	0	0	1

Figure 2: Base de données binaire $\mathcal{D}'$

Dans un environnement massivement distribué et avec un très grand volume des données, la découverte des motifs informatifs maximaux de taille k (*miki*) présente un grand défi. Les approches conventionnelles qui ont été proposées pour les environnements centralisés devraient être soigneusement conçues pour être parallélisées. Cependant, dans la littérature, il n'y a pas des solutions proposées pour l'extraction des *miki* en parallèle. Dans cette section, nous limitons notre discussion à l'algorithme ForwardSelection [23].

Parallel ForwardSelection Algorithm: Comme l'algorithme ForwardSelection utilise une approche par-nivaux pour déterminer les *miki* de taille k, une version parallèle de cet algorithme aurait besoin de k jobs. Alors, avec un très grand volume des données et une grande valeure de k, les performances de ForwardSelection se dégrade. Ces performances sont du aux accès multiples à la base des données, la génération des *miki* candidats et la phase de comparaison des valeurs des entropies dans le reducer. En outre, une version parallèle de ForwardSelection pourrait souffrir d'autres limitations. En particulier, le taux des données échangées entre les mappers et les reducers serait très grand ce qui impacte la performance globale du processus d'extraction des *miki*.

Split	A	B	C	D
S_1	1	0	1	0
	1	0	1	0
	1	0	0	0
	1	1	0	0
	1	1	1	0
S_2	0	0	1	0
	0	0	0	0
	0	0	0	1

Figure 3: Partitions des Données

Example 5 *Considérons la base de données $\mathcal{D}'$ illustrée par la Figure 2. Supposons que nous voulons déterminer les miki de taille k égale à 2 en utilisant une version parallèle de* ForwardSelection. *Supposons que la base de données $\mathcal{D}'$ est divisée en deux partitions (splits) comme illustré par la Figure 3. Chaque partition des données (respectivement S_1 et S_2) est traitée par un mapper (respectivement m_1 and m_2). Dans le premier job, chaque mapper traite sa partition des données et envoie chaque item comme clé et sa projection (i.e., combinaison des '0s' et '1s') comme valeur. Par exemple, m_1 envoie (A, 1) 5 fois au reducer. m_2 envoie (A, 0) 3 fois au reducer (une simple optimisation peut être utilisée consiste à envoyer seulement les items qui apparaissent dans les transactions i.e., avec projections des'1s'). Puis, le reducer prend en charge le calcul des entropies des items et détermine l'item qui a l'entropie la plus forte. Dans une deuxième job, l'item avec l'entropie la plus forte est combinée avec chaque item restant dans la base de donnéespour construire des miki candidats de taille k égale à 2. Ensuite, la même processus est lancé pour déterminer le miki de taille 2. Dans cet exemple, le résultat de premier job sera {C} ($H(C) = 1$) et le résultat de second job sera {C A} ($H(CA) = 1.905$). Ce processus d'extraction des miki continue jusqu'à la détermination des miki de taille k i.e., en utilisant k jobs de MapReduce. Cependant, cette approche est très coûteuse, en particulier quand k tend vers des grandes valeurs et le volume des données est grand.*

Les algorithmes parallèles de classification de données

Plusieurs solutions ont été proposées pour résoudre les problèmes de la classification supervisée de données [24], [25], [26], [27], [28] and [29]. Bien que les alternatives de la littérature soient bien capables d'effectuer une tâche de classification complète et exacte, chacune des approches possède ses limites.

Parmi les algorithmes nous citons l'algorithme K-nearest neighbor (Knn) dont les performances sont linéaires par rapport à la taille des données d'apprentissage en entrée. Cependant, avec un nombre très élevé d'instances déjà étiquetées et classées, l'algorithme s'avère très coûteux en termes de temps d'exécution pour classer une nouvelle instance en essayant de calculer toutes les distances possibles par rapport à la nouvelle instance.

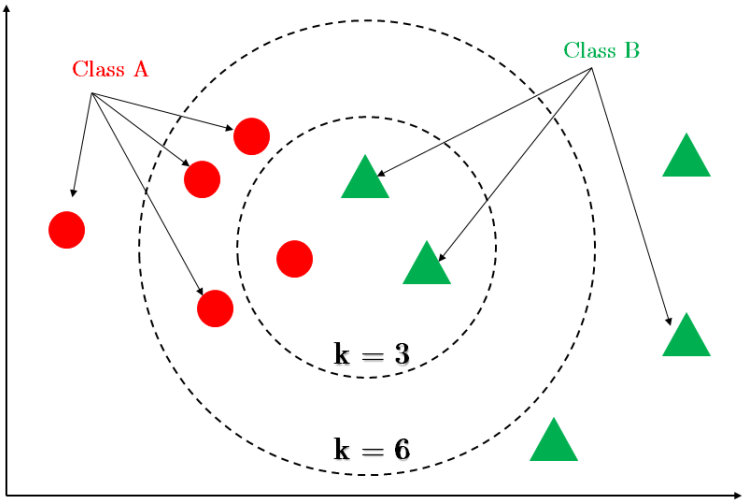


Figure 4: La classification selon l'algorithme Knn

L'algorithme Rocchio [29] a été largement utilisé pour la classification du texte [30]. Il est un des plus vieux algorithmes de classification destiné à l'amélioration des systèmes de recherche documentaires. L'avantage de ce type de classifieurs est la simplicité et l'interprétabilité. Pour un expert, ce profil prototype est plus compréhensible qu'un réseau de neurones par exemple. L'apprentissage de ce type de classifieur est souvent précédé par une sélection et une réduction de termes. Ce classifieur s'appuie sur une représentation vectorielle des documents. Malgré son simple mode d'exécution et sa rapide tâche de classification, l'algorithme affiche une faible précision de classification.

L'outil arbre de décision (AD) [31] manifeste de meilleurs performances en classification de données comparé aux algorithmes Rocchio et Knn. En effet, prenons l'exemple d'une classification textuelle en utilisant l'outil arbre de décision, chaque terme (mot, motifs, attribut, etc) du corpus sera attribué à une feuille de l'arbre. Chaque branche de l'arbre est pondérée selon le poids du terme dans le texte. Les feuilles (représentant les noeuds de l'arbre) seront étiquetées par des catégories spécifiques (les classes). Afin de construire son

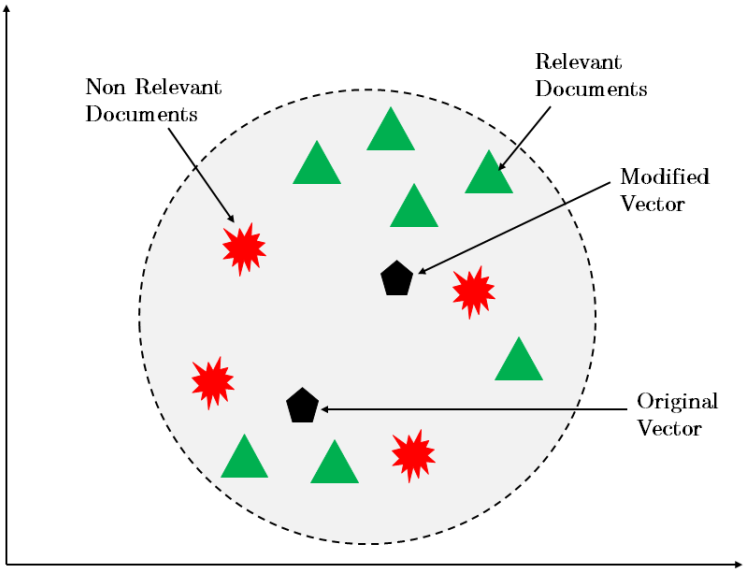


Figure 5: La classification selon l'algorithme Rocchio

modèle, l'outil arbre de décision utilise la stratégie "diviser pour régner", un processus de construction qui se repose sur un élagage du corpus basé essentiellement sur l'imposition d'une métrique statistique et d'heuristiques introduites (poids des termes dans le texte). Néanmoins, AD permet de générer des modèls d'arbres très volumineux et complexes, difficiles à manipuler causant un coût remarquable en termes de temps d'exécution.

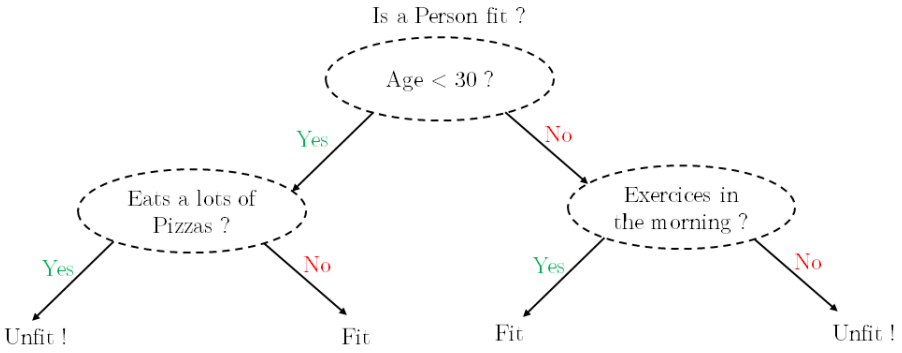


Figure 6: La classification selon l'approche Arbre de Decision

Dans [32] les auteurs détaillent concrètement l'approche Support Vector Machine (SVM). L'approche SVM représente des algorithmes d'apprentissage initialement construits pour la classification binaire. L'idée est de rechercher une règle de décision basée sur une séparation par hyperplan de marge optimale, méthode relativement récente qui découle de premiers travaux théoriques de Vapnik et Chervonenkis en 1995, démocratisés à

partir de 2000 [33]. Une fois que la phase d'apprentissage est terminée (les données sont rapportées dans un espace à multiples dimensions), il s'agit alors de définir dans cet espace un plan permettant de séparer deux groupes de sujets. Après avoir dessiné la frontière, le SVM est maintenant capable de prédire à quelle catégorie appartient une instance qu'il n'avait jamais vue auparavant. Même si SVM a montré d'excellentes performances, son plus grand défaut réside sur le nombre très élevé de classe à gérer. La Figure 7 illustre un exemple de classification SVM.

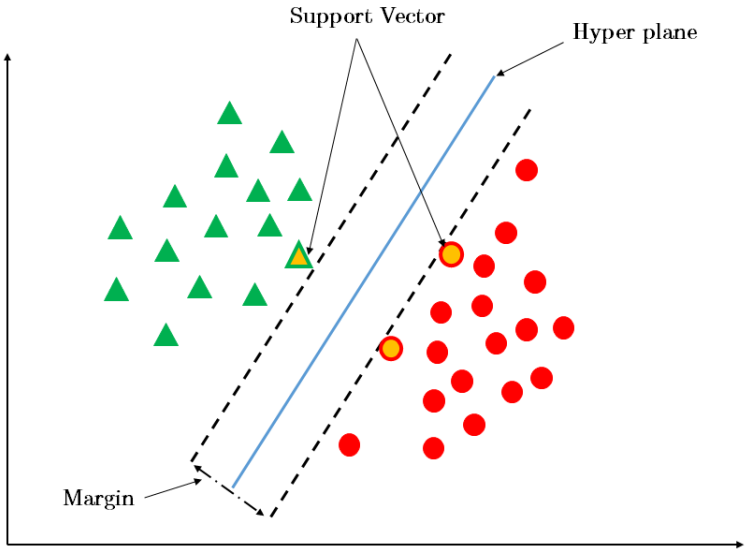


Figure 7: La classification selon l'approche SVM

La méthode de classification naïve bayésienne [34] est un algorithme d'apprentissage supervisé (supervised machine learning), qui permet de classifier un ensemble d'observations selon des règles déterminées par l'algorithme lui-même. Cet outil de classification doit dans un premier temps être entrainé sur un jeu de données d'apprentissage qui montre la classe attendue en fonction des entrées. Pendant la phase d'apprentissage, l'algorithme élabore ses règles de classification sur ce jeu de donnée, pour les appliquer dans un second temps à la classification d'un jeu de données de prédiction. Le classifieur bayésien naïf implique que les classes du jeu de données d'apprentissage soit connu et fourni, d'où le caractère supervisé de l'outil.

Historiquement, la classification naïve bayésienne fut utilisée pour la classification de documents et l'élaboration de filtres anti-spam. Aujourd'hui, c'est un algorithme renommé dont les applications peuvent être rencontrées dans de nombreux domaines. Parmi ces atouts les plus significatifs, on citera son apprentissage rapide qui ne nécessite pas un gros volume de données et son extrême rapidité d'exécution comparé à d'autres méthodes plus complexes. Finalement, malgré la forte hypothèse simplificatrice d'indépendance des variables, la classification naïve bayésienne obtient de bons résultats dans de nombreuses

applications de la vie courante. A la base de la classification naïve bayésienne se trouve le théorème de Bayes avec l'hypothèse simplificatrice, dite naïve, d'indépendance entre toutes les paires de variables. Afin d'ameliorer encore l'approche, de nouvelles améliorations ont été appliquées pour produire l'approche de classification bayésienne multinomial.

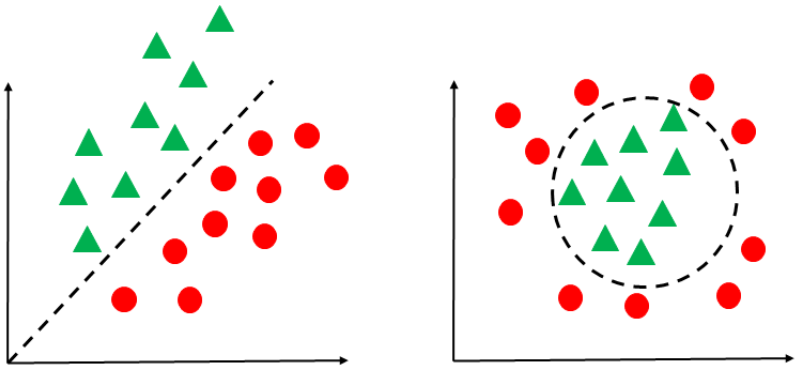


Figure 8: Exemples simples de la classification selon la méthode de classification naïve bayésienne

En général, les principaux défis dans les problèmes de classification ont été liés à l'accélération du temps d'exécution et la précision des modèles. Une technique intéressante appelée "Ensemble of Classifier" (EC en court) [35] a été proposée pour améliorer la précision de la classification d'une nouvelle instance. L'idée est simple et très élégante. Sur la base d'un ensemble de modèles de base (c'est-à-dire plusieurs classifieurs), il est décidé de classer une instance anonyme. Cette technique a été appliquée avec succès avec l'algorithme Random Forest (RF) [36]. Au lieu de construire un modèle à arbre unique, un ensemble d'arbres est construit. Pour classer une nouvelle instance, les décisions de toutes les arbres de la forêt sont prises en compte. La technique EC a abouti à une amélioration significative de la précision de plusieurs approches de classification conventionnelles. De plus, EC a offert la possibilité de combiner facilement différentes techniques de classification. Toutefois, EC a rencontré certains problèmes, notamment liés à la construction des modèles d'apprentissage. La figure 9 illustre un exemple d'application de la technique "Ensemble of Classifier" avec l'algorithme foret d'arbre décisionnels (Random Forrest).

Avec la technique EC, une bonne précision de classification peut être obtenue. Le principal problème se révèle être le temps de réponse de l'ensemble du processus d'apprentissage et de classification appliqués dans de très grandes bases de données. A cette fin, des approches parallèles ont été proposées pour les techniques de classification de texte [37], [36]. Par exemple, RF a été parallélisé dans [38]. Cependant, avec un nombre élevé d'attributs, la consommation mémoire devient un problème, là où la taille des arbres ne pourra pas être stockée.

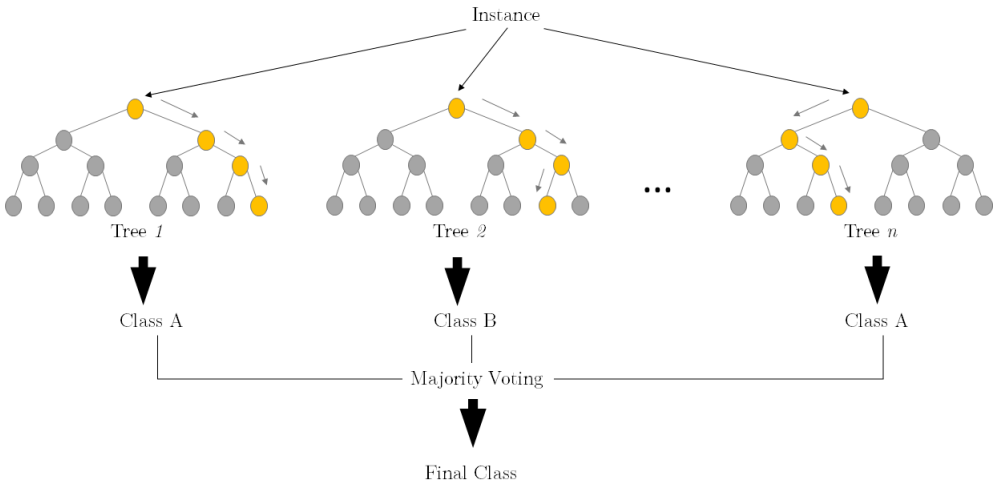


Figure 9: La classification selon la méthode forêt d'arbres décisionnels

Contributions

L'objectif de cette thèse est de développer des nouvelles techniques pour l'extraction parallèle des motifs fermés fréquents et *miki* dans des environnements massivement distribués. Nos contributions majeures sont comme suit.

Extraction Rapide des Motifs Fermés Fréquents. Dans ce travail, nous proposons DCIM (Distributed-Closed-Itemsets-Mining), un algorithme parallèle pour l'extraction des motifs fermés fréquents. DCIM rend le processus d'extraction des motifs fermés fréquents dans les données massives (Des dizaines de gigaoctets de données) simple et compact. DCIM fouille une telle base de données en deux jobs MapReduce, ce qui réduit significativement le temps d'exécution, le coût de communication des données et la consommation énergétique dans les plates-formes de calcul distribuées. En se basant sur une méthode de partitionnement des données nommée Item Based Data Partitioning, DCIM fouille chaque partition des données d'une façon indépendante en utilisant un seuil de support minimum relatif. De plus, DCIM introduit une nouvelle approche de modélisation de données qui permet de codifier la base transactionnelle en nombres premiers, c'est ainsi que nous exploitons les différentes propriétés liées aux nombres premiers pour une extraction efficace des motifs fermés fréquents. Notre approche DCIM a été largement évaluée sur des grands volumes des données du monde réel. Nos différent résultats confirment l'efficacité de notre approche.

Extraction massive des *miki* dans le Streaming. Dans ce travail, nous adressons le problème d'extraction des *miki* en parallèle basé sur l'entropie à partir d'un flux

continu de données. Nous proposons PentroS (Parallel entropy computing over Streas), un algorithme parallèle pour l'extraction des *miki* en utilisant le framework Spark streaming. Avec PentroS, nous fournissons plusieurs techniques d'optimisations de calcul de l'entropie ainsi que des propositions pour l'optimisation du taux de latence et des mise à jour des données entrantes et sortantes du flux, pour améliorer l'extraction des *miki*. Ces différents techniques réduisent énormément le temps d'exécution, le taux de communication des données, la consommation énergétique dans les plates-formes de calcul distribué et augmente la quantité des données à traiter sur un batch du flux. Nous avons évalué la performance de PentroS sur des données massives de monde réel. Nos expérimentations confirment l'efficacité de notre approche pour extraire les *miki*.

Classification parallèle de données. Dans ce travail, nous abordons le problème de la classification parallèle dans des environnements hautement distribués. Nous proposons Eec (Ensemble of Ensembles of Classifiers) pour rendre la tâche de la classification des documents (Objets, transactions, etc.) simple et très efficace. L'algorithme est composé de deux Jobs Spark. Combinant plusieurs classifieurs, Eec exploite profondément le parallélisme sous le framework Spark pour non seulement réduire le temps d'exécution mais aussi améliorer de manière significative la précision de la classification en effectuant deux étapes de prise de décision. Nous montrons que la précision de la classification de Eec a été améliorée en utilisant les motifs informatifs et que l'erreur de classification peut être limitée à une petite valeur dans un environnement parallèle. Eec a été largement évalué en utilisant "English Wikipedia articles" et "clue Web" datsets. Nos résultats expérimentaux montrent que Eec est significativement plus efficace et précis que les autres alternatives de la littérature.

Publications

- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. Maximally Informative k-Itemset Mining from Massively Distributed Data Streams. In ACM Symposium on Applied Computing "ACM SAC 2018", April 2018.

- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. Massively Distributed Environments and Closed Itemset Mining: The DCIM Approach. In Conference on Advanced Information Systems Engineering 2017 "CAiSE 2017" on pages 231-246, Jun 2017.

- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. A Prime Number Based Approach for Closed Frequent Itemset Mining in Big Data. In International Conference on Database and Expert Systems Applications 2015 "DEXA 2015" on pages 509-516, Sept 2015.

Organisation de la thèse

La suite de cette thèse est organisée comme suit.

Dans le chapitre 2, nous révisons des notions préliminaires liées à la fouille des motifs. La section 2.2 présente les définitions de base relatives à la recherche d'itemsets dans les bases de données. Ensuite, dans la section 2.3, nous détaillons les paramètres mathématiques issue de l'analyse des concepts formels liés au processus de recherche des itemsets fermés fréquents. La troisième section 2.4 sera dediée à l'introduction des éléments de la théorie de l'information pour formuler le cadre d'extraction des motifs informatifs à partir des bases de données.

Dans le chapitre 3 nous présentons une étude critique des travaux de l'état de l'art. Ce chapitre est divisé en trois sections: Dans la Section 3.2, nous étudions les techniques utilisées dans la littérature pour la découverte et l'exploration des connaissances dans les environnements centralisés. En particulier, nous focalisons notre étude sur deux sujets: l'extraction des motifs fermés fréquents et l'extraction des motifs informatifs maximaux de taille k. Dans la Section 3.3, nous introduisons les techniques et les méthodes récentes qui ont été proposées pour traiter les données massives. Dans la Section 3.4, nous étudions les approches parallèles qui ont été proposées dans la littérature pour l'extraction des motifs fermés fréquents qui feront l'objet du chapitre 4 et la découverte des motifs informatifs maximaux de taille k qui sera le thème du chapitre 5.

Dans le chapitre 4, nous adressons le problème d'extraction des motifs fermés fréquents. Dans la Section 4.3, nous proposons l'algorithme DCIM et nous expliquons son principe de fonctionnement. Dans la Section 4.4, nous effectuons des expérimentations différentes pour évaluer notre approche en utilisant des données massives du monde réel. Finalement, dans la Section 4.5, nous résumons et concluons notre travail.

Dans le chapitre 5, nous étudions le problème d'extraction des motifs maximaux informatifs de taille k à partir d'un flux continu de données. Dans la Section 5.4, nous proposons PentroS, notre algorithme parallèle pour l'extraction des *miki*. Dans la Section 5.5, nous validons notre approche en simulant un flux à partir des données massives de monde réel. Dans la Section 5.6, nous concluons notre travail.

Dans le chapitre 6, nous adressons le problème de la classification supervisée de données en utilisant les motifs informatifs. Dans la section 6.3, nous discutons les travaux de l'état de l'art. Dans la section 6.5, nous proposons notre algorithme Eec pour la calssification parallèle des données. Dans la section 6.6, nous validons notre algorithme en effectuant multiples expérimentations en utilisant des données massives du monde réel. Section 6.7 conclut ce chapitre.

Conclusion

Dans cette thèse, nous avons abordé deux problème principaux: l'extraction parallèle des motifs fermés fréquents à partir d'une Big Data et l'extraction parallèle des motifs infor-

matifs maximales de taille k à partir d'un flux continu de données. Dans ce chapitre, nous avons discuté les problèmes reliés au processus d'extractions des motifs fermés fréquents et celles des *miki*, en étudions les approches proposées dans l'état de l'art. Le problème majeur et les limitations de ces processus d'extraction des motifs sont reliés particulièrement aux accès multiples à la base de donnéeset la capacité de la mémoire. Typiquement, ces différents limitations présentent un défi majeur quand le volume des données est grand et le support minimum est très petit ou la taille k de *miki* est grand.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"With Great Power, Comes Great Responsibilities"
Benjamin Parker

1.1 Context

In the past years, advances in hardware and software technologies have made it possible to produce increasing amounts of diverse data, rapidly growing over time and needing more space and computing power. These data contain a large amounts of hidden and valuable knowledge that can be hardly exploited, calling for adequate knowledge discovery approaches and tools. The storage of these large amounts of data is less challenging than their processing.

Being a historical term (2238 BC), knowledge discovery has converged throw multiple generations. In 1662, Jhon Graunt [1] published his book analyzing the mortality rate in London and tried to predict the Bionic plague. In 1763, Thomas Bayes [2] showed that we can fix not only probabilities from the observations of an experiment, but also the parameters relating to these probabilities. It was not until the late 80s that Rakesh Aggrawel used the term to mine datasets with the size of 1MB.

Nowadays, the process of knowledge discovery is defined as an important actor to illustrate knowledge from extremely massive and labyrinthed databases. To handle such

large data, one solution is to distribute them on multiple machines and process them in parallel. In data mining, this solution requires a profound revision of the different existing techniques and algorithms.

Data mining [39] wraps a set of methods and techniques that allow analyzing and exploring the large amounts of data. Frequent Itemset Mining ($\mathcal{FIM}$) presents a variant of these techniques with the aim of determining the itemsets (features, patterns, or terms) which frequently co-occur together in data. The co-occurrence frequency is a measure of informativeness (*i.e.* interestingness) which helps to measure the utility of the itemsets based on their number of co-occurrences in data.

$\mathcal{FIM}$ has a large range of applications in various domains. For instance, in text mining [40], as it will be better illustrated in chapter 3 of this thesis, $\mathcal{FIM}$ can be used to determine the co-occurrence number of the words in a very large database. In e-commerce [41], $\mathcal{FIM}$ can be utilized to recommend products such as books, clothing, etc.

In practice, the number of frequent itemsets can be overwhelmingly large, hence hampering their effective exploitation by the end-users. In order to reduce the number of mined frequent itemsets, statistical measures are of common use. Nevertheless, if the minimal support threshold is set too low or data are highly correlated, no matter how efficient the frequent pattern mining algorithm is, generating all the frequent itemsets is impossible. Moreover, the set of itemsets presents redundancy in the sense that many itemsets convey the same information [42]. To overcome this problem, several proposals have been made to only build a manageable-sized set of patterns from which we can generate all the frequent patterns along with their exact frequencies. Such a reduced set is better known as an *exact concise (or condensed) representation*. A concise representation only stores a non redundant cover of all frequent patterns. In many practical situations, this cover is considerably smaller than the complete collection of all frequent patterns. Therefore, a concise representation can be used in those situations where it is impossible or inefficient to get out all the frequent patterns.

Beyond high compactness rates, an exact concise representation makes it possible to guess the frequency status of an itemset and to exactly retrieve its exact support when an itemset is (potentially) interesting w.r.t. statistical measures. Many exact concise representations of frequent patterns have been thus proposed in the literature [7, 43, 44, 45]. The ones based on closed itemsets [7] have had a large interest since their proposal. In this thesis, we focus, in its first part, on Closed Frequent Itemsets ($\mathcal{CFI}$). Many $\mathcal{CFI}$ algorithms have been proposed to mainly reduce the huge number of mined frequent patterns.

However, for some specific application domains, the co-occurrence frequency measure fails to capture and determine all interesting or informative itemsets in data. This is particularly the case when data are sparse and calling for large-scale distribution. To this end, other informativeness measures should be taken into account. One of these interesting measures is the joint entropy of itemsets. In particular, itemsets with maximum joint entropy would be informative; *i.e.* the items that constitute such an informative

itemset have a weak relationship between each other, but together maximally shatter the data. The informative itemsets based on joint entropy measure are of significant use in various domains. For instance, in classification, among all available features (*i.e.* independent attributes), we always prefer a small subset of features (featureset or itemset) which contains highly relevant items to the classification task. A maximally informative k-itemset (*miki*) is the informative itemset of size k that has maximum joint entropy.

Recently, with the availability of powerful programming models such as MapReduce [9] and Spark [10], the processing of massive amounts of data has become handy. However, most of the parallel data mining algorithms still suffer from several drawbacks.

Parallel $\mathcal{FIM}$ algorithms have brought the same limitations as in their sequential versions. These limitations have been primarily related to their core mining principle. For instance, the centralized (*i.e.* sequential) implementation of the popular Apriori [5] algorithm for $\mathcal{FIM}$, requires repeated disc access. Likewise, a parallel version of the Apriori algorithm would require multiple scans of the database, thus multiple parallel jobs. Although the FP-Growth algorithm [46] has been considered as one of the most powerful $\mathcal{FIM}$ algorithms, with very low minimum support and very large amount of data, its parallel version, PFP-GROWTH [12] cannot scale due to memory issues. Up to our knowledge, at the beginning of our work, there have been no algorithms developed under the MapReduce framework that tackles the problem of mining $\mathcal{CFI}$s. We will depict in chapter 4 some new alternatives that have been proposed recently.

Similarly, the parallel mining of the itemsets based on joint entropy as an informativeness measure does not escape the rule from suffering from various drawbacks as for $\mathcal{FIM}$. Mining *miki* in parallel is far from being a trivial task. Moreover, mining it in a dynamic aspect of data (over data streaming) is more difficult and very hard. This is because the parallel computation of the joint entropy is costly due to the high access to the disc. For instance, a parallel version of the popular *ForwardSelection* [23] algorithm would require several parallel jobs to determine *miki* over a huge amount of incoming and outgoing batches of data.

In addition to the regular issues that a parallel data mining algorithm may have when processing massive amounts of data, in massively distributed environments, the quantity of transferred data may impact the whole mining process. Therefore, a careful parallel design of these algorithms should be considered.

Let us illustrate the potential issues and problems that may happen for a parallel mining algorithm when processing very large amounts of data through the following examples.

Example 6 *Suppose we are given a very low minimum support, and we would like to determine the frequent closed itemsets in a very large database $\mathcal{D}$ using a parallel version of the popular Apriori-close algorithm. The required number of MapReduce jobs would be proportional to the size of the most lengthy candidate itemset. In a massively distributed environment, this would result in a very poor performance since the transferred data (e.g. candidate itemsets) between mappers and reducers would be very high. Consider a parallel version of the FP-Close Algorithm for mining the database $\mathcal{D}$. With a very low minimum*

support and an exhaustive search of $\mathcal{CFI}$ (i.e. parameter k set to infinity), the algorithm would suffer from various limitations. First, the frequent-pattern-tree may not fit into the memory. Second, if it is not the case, the transferred data would be very high, which would highly impact the mining process.

Example 7 *Suppose we would like to determine the miki in parallel. Consider a parallel version of the popular ForwardSelection algorithm. Depending on size k of the miki to be discovered, the algorithm would perform k MapReduce jobs. This would result in a very poor performance. Besides the multiple database scans, the number of itemset candidates would be very high. Thus, the parallel mining of miki falls in the same limitations and restrictions of those of the parallel mining of frequent itemsets.*

On the other hand, classification [14] is one of the building bricks in data mining and information retrieval. As a matter of fact, classification is a supervised learning process that consists of an automatic assignment of an instance to one (*i.e.* single-label classification) or more (*i.e.* multi-label classification) predefined categories, *e.g.* classes, target attributes or dependent attributes. The classification process turns out to learn a system (*i.e.* model or classifier) which is capable of making good decisions based on its past experience.

Shortly, the classification problem can be defined as follows [14]. Given a training dataset with a fixed number of labeled instances (i.e., each instance has been already assigned to a predefined category), a model will be built which can classify a new unseen instance to an appropriate category with small classification error.

Nowadays, as mentioned previously, we are completely overwhelmed with data coming from different sources such as social networks and sensors. To process these large volumes of data, conventional classifier algorithms have shown their limitations. Typically, data cannot fit into memory, so a classifier cannot learn from large datasets. In addition, classification algorithms are no longer able to efficiently handle large amounts of data in centralized environments.

In this thesis, we address the problem of parallel classification in highly distributed environments. We propose Ensemble of Ensembles of Classifier (Eec), a parallel, scalable and highly accurate classifier algorithm. Eec renders a classification task simple, yet very efficient. Its working process is made up of two simple and compact Spark jobs. Calling to more than one classifier, Eec cleverly exploits the parallelism setting not only to reduce the execution time but also to significantly improve the classification accuracy by performing two level decision making steps. We show that the Eec classification accuracy has been improved by using informative features and the classification error can be bounded to a small value. Eec has been extensively evaluated using various real-world, large datasets. Our experimental results suggest that Eec is much more efficient and accurate than alternative approaches.

1.2 Contributions

The objective of this thesis is to develop new techniques for parallel mining of frequent closed itemsets and *miki* in massively distributed environments and to develop a parallel supervised classification algorithm using *miki* as a new approach for feature selection. Our main contributions are as follows.

Fast Parallel Mining of Frequent Closed Itemsets in MapReduce. In this work, we study the effectiveness and leverage of specific data transformation strategies for improving the parallel frequent closed itemset mining performance in MapReduce. By offering a clever data transformation and an optimal organization of the extraction algorithms, we show that the itemset discovery effectiveness does not only depend on the deployed algorithms. We propose DCIM, a solution for fast mining of frequent closed itemsets in MapReduce. Our method allows discovering itemsets from massive datasets, where standard solutions from the literature do not scale. Indeed, in a massively distributed environment, the arrangement of both the data and the different processes can make the global job either completely inoperative or very effective. Our proposal has been evaluated using real-world datasets and the results illustrate a significant scale-up obtained with a very low minimum support, which confirms the effectiveness of our approach.

Fast Parallel Mining of Maximally Informative k-Itemsets over Data Stream. In this work, we address the problem of parallel *miki* mining based on joint entropy. We propose PENTROS a highly scalable, parallel *miki* mining algorithm. With PENTROS, we provide a set of significant optimizations for calculating the joint entropy of *miki* having different sizes over data streams, which drastically reduces execution time, communication cost and energy consumption and remarkably increase the throughput of batches with a huge amount of incoming and outgoing data, in a distributed computational platform. PENTROS is extensively evaluated using massive real-world datasets. Our experimental results confirm the effectiveness of our proposal by the significant scale-up obtained with lengthy itemsets and over very large data streams.

Fast Parallel Ensemble of Ensembles of classifiers. In this work, we address the problem of parallel classification in highly distributed environments. We propose EEC, a parallel, scalable and highly accurate classifier algorithm. EEC renders classification task simple, yet very efficient. Its working process comprises two simple and compact jobs. Calling to more than one classifier, EEC cleverly exploits the parallelism setting not only to decrease the execution time but also to significantly improve the classification accuracy by carrying out two level decision making steps. We demonstrate that the EEC classification accuracy is improved by utilizing informative itemsets and that the classification error can be bounded to a small value. EEC is extensively evaluated using various real-world, large datasets. Our experimental results suggest that EEC is significantly more efficient and accurate than alternative approaches.

1.3 Publications

- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. A Prime Number Based Approach for Closed Frequent Itemset Mining in Big Data. In International Conference on Database and Expert Systems Applications 2015 "DEXA 2015" on pages 509-516, Sept 2015.
- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. Massively Distributed Environments and Closed Itemset Mining: The DCIM Approach. In Conference on Advanced Information Systems Engineering 2017 "CAiSE 2017" on pages 231-246, Jun 2017.
- Mehdi Zitouni, Reza Akbarinia, Sadok Ben Yahia and Florent Masseglia. Maximally Informative k-Itemset Mining from Massively Distributed Data Streams. In ACM Symposium on Applied Computing "ACM SAC 2018", April 2018.

1.4 Road Map

The rest of the thesis is organized as follows.

In chapter 2, we introduce the preliminary notions that will be of use in this thesis. In section 2.2, we sketch the basic definitions for mining itemsets in databases. Section 2.3 details some mathematical foundations issued from the Formal Concept Analysis (FCA) bases. $\mathcal{CFI}$ as a concise representation of frequent itemsets is based on large proposals issued from FCA. Section 2.4 will be dedicated to introduce elements of information theories as part of the *miki* mining problem from datasets.

In chapter 3, we review the state of the art. It is split into three main sections: In section 3.2, we provide a general overview on the main knowledge discovery techniques in a centralized environment. In particular, we deal with two techniques: frequent itemset mining, and maximally informative k-itemset. In section 3.3, we introduce the cutting-edge solutions and techniques that are used to process massive amounts of data. In section 3.4, we deal with the basics, recently used parallel techniques for discovering knowledge from large databases. Specifically, we focus on two problems: Parallel $\mathcal{CFI}$ mining, which will be the subject of chapter 4 and parallel *miki* mining, which will be the focus of chapter 5.

In chapter 4, we address the problem of $\mathcal{CFI}$ mining in very large databases. In section 4.3, we propose our Dcim algorithm and we thoroughly explain its mining principle. In section 4.4, we validate our proposal through extensive, different experiments using very large real-world datasets. Eventually, in section 4.5, we conclude our work.

In chapter 5, we deal with the problem of mining *miki* in big data. In section 5.4, we introduce our PentroS algorithm for *miki* parallel discovery. We thoroughly detail its mining principle. In section 5.5, we validate our approach by carrying out various, extensive experiments using very massive real-world data streams. Finally, in section 5.6, we summarize our work.

In chapter 6, we tackle the problem of parallel classification, in which we sketch a use case for *miki* in a supervised classification process as a new approach for feature selection. In section 6.3, we discuss the related work and multiple alternatives from the litterature. In section 6.5, we propose our *Eec* algorithm and we depict its core working process. Section 6.6 reports on our experimental evaluation over various real-world datasets. Section 6.7 concludes the chapter.

Chapter 2

Preliminary Notions

2.1 Introduction

Many significant and hidden relations between data can be exploited from a dataset. Such relations can be useful for end users, like domain experts and decision makers. These latter can exploit them for various objectives aiming at improving their decision quality.

In this chapter, we present the problem of knowledge discovery based on frequent itemsets and informative itemsets. We also recall the mathematical background of Formal Concept Analysis (FCA) and element of information theory.

The organization of the chapter is as follows : Section 2.2 presents the basic definitions related to itemsest search in datasets. Afterwards, in section 2.3 we detail the FCA mathematical settings related to itemset search process. Section 2.4 sketches the elements of information theory, related to the definition of itemsets with high entropy, a new measure for the informativeness of itemsets.

2.2 Itemset Search Space

This section presents some basic definitions that will be useful in the remainder.

2.2.1 Extraction Context and Itemsets

In this thesis, we will consider datasets represented using binary contexts defined as follows.

Definition 4 *(Extraction Context)*
An extraction context (or context in short or database) is a triplet $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{M})$, where $\mathcal{O}$ is a finite set of objects (or transactions), $\mathcal{I}$ is a finite set of items (or attributes) and $\mathcal{M}$ is a binary (incidence) relation (i.e., $\mathcal{M} \subseteq \mathcal{O} \times \mathcal{I}$). A couple (o,i) $\in \mathcal{M}$ if the object $o \in \mathcal{O}$ has the item $i \in \mathcal{I}$.

Example 8 *Consider the context $\mathcal{D}$ given in Figure 2.1, used as a running example trough this chapter. Here, $\mathcal{O}$ ={1, 2, 3, 4, 5, 6, 7} and $\mathcal{I}$ = {A, B, C, D, E}. The couple (3, E) $\in \mathcal{M}$ since it is crossed in the matrix, on the contrary of the couple (4, C) whose associated cell is not crossed in the matrix.*

	A	B	C	D	E
1			×	×	×
2		×	×		×
3	×	×	×		×
4		×			×
5	×	×		×	
6	×	×	×		×
7		×	×	×	×

Figure 2.1: Database $\mathcal{D}$

An itemset is a set of items. For example, {A, B, C, E} is an itemset composed of the items A, B, C and E. In the remainder, we use a separator-free form for the sets; *e.g.* CDE stands for the itemset {C, D, E}. The terms *dataset*, *database* and (*extraction*) *context* are also used interchangeably throughout the remainder of the thesis. It is the same for *transactions* and *objects*.

2.2.2 Itemset Supports: Links and Associated Constraints

For each itemset in a context, there are different kinds of support. In the following definition we detail each kind of itemset support.

Definition 5 *(Itemset Support)*
Let $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{M})$ be an extraction context. We distinguish three kinds of support associated to a non-empty itemset I:

- ***Conjunctive support:*** $Supp(\wedge I) = |\{ o \in \mathcal{O} \mid (\forall\, i \in I, (o,i) \in \mathcal{M})\}|$,
- ***Disjunctive support:*** $Supp(\vee I) = |\{ o \in \mathcal{O} \mid (\exists\, i \in I, (o,i) \in \mathcal{M})\}|$, *and,*
- ***Negative support:*** $Supp(\overline{I}) = |\{ o \in \mathcal{O} \mid (\forall\, i \in I, (o,i) \notin \mathcal{M})\}|$.

Roughly speaking, the different kinds of support are defined as follows:

- **Supp($\wedge I$)** is the number of transactions containing all items of I. In this case, I can be seen as a conjunction of items (i.e. $i_1 \wedge i_2 \wedge \cdots \wedge i_n$) such that the appearance of one of its items is conditioned by the appearance of all remaining ones to take into consideration that I satisfies a given transaction.

- **Supp($\vee I$)** is the number of transactions containing at least one item of I. In this case, I can be seen as a distinction of items (i.e. $i_1 \vee i_2 \vee \cdots \vee i_n$) such that the presence of one of its items in a transaction is sufficient to satisfy it independently the remaining items.
- **Supp($\overline{I}$)** is the number of transactions that do not contain any item of I. In other words, it contains the respective negations of all items of I (i.e. $\overline{i_1} \wedge \overline{i_2} \wedge \cdots \wedge \overline{i_n}$).

Example 9 *Consider context $\mathcal{D}$ from Figure 2.1. The different supports that can be associated to the itemset CD are: Supp($\wedge CD$) = 2, Supp($\vee CD$) = 6 and Supp($\overline{CD}$) = 1.*

Note that the conjunctive support of the empty set is equal to $\mid \mathcal{O} \mid$ since it is included in all transactions (objects). While the disjunctive support is not defined on this pattern since it does not contain any item.

The next proposition summarizes important properties related to itemset support.

Proposition 1 *Let $i \in \mathcal{I}$ and $P, Q \in \mathcal{I}$. The following properties hold:*

- *Supp($\wedge i$) = Supp($\vee i$).*
- *Supp($\wedge P$) $\leq$ Supp($\vee P$) for $P \neq \emptyset$.*
- *If $P \subseteq Q$, then Supp($\wedge P$) $\geq$ Supp($\wedge Q$).*
- *If $P \neq \emptyset$ and $P \subseteq Q$, then Supp($\vee P$) $\leq$ Supp($\vee Q$).*

Given the respective disjunctive supports of the susets of an arbitary itemset, we are able to derive its conjunctive support using *inclusion − exclusionidentities* [47]. Furthermore, thanks to the *DeMorgan*'s law, we can to straightforwardly derive its negative support. Lemma 1 depicts these equations.

Lemma 1 *Let $Q \subseteq \mathcal{I}$ be an arbitrary itemset and $P \subset Q$. Q's conjunctive support and negative support are respectively derived as follows [47]:*

$$Supp(\wedge Q) = \sum_{\emptyset \subset P \subseteq Q} (-1)^{|P|-1} Supp(\vee P) \tag{2.1}$$

$$Supp(\overline{Q}) = \mid \mathcal{O} \mid - Supp(\vee Q) \tag{2.2}$$

Example 10 *Consider the context of Figure 2.1. Given the respective disjunctive support of AC subsets, its conjunctive support and negative support are inferred as follows:*

$$\begin{aligned} Supp(\wedge AC) &= (-1)^{|AC|-1} Supp(\vee AC) + (-1)^{|A|-1} Supp(\vee A) + (-1)^{|C|-1} Supp(\vee C) \\ &= -Supp(\vee AC) + Supp(\vee A) + Supp(\vee C) = -6 + 3 + 5 = 2. \\ Supp(\overline{AC}) &= \mid \mathcal{O} \mid - Supp(\vee AC) = 7 - 6 = 1 \end{aligned}$$

To prune the search space of itemsets, different types of constraints are investigated. Anti-monotone and monotone constraints, defined in the following, are the most used ones [48].

Definition 6 *(Anti-monotone constraint)*
Let $I \in \mathcal{I}$. A constraint ϕ is said to be monotone if $\forall I_1 \subseteq I$; I satisfies $\phi \rightarrow I_1$ satisfies ϕ.

Definition 7 *(Monotone constraint)*
Let $I \in \mathcal{I}$. A constraint ϕ is said to be monotone if $\forall I_1 \supseteq I$; I satisfies $\phi \rightarrow I_1$ satisfies ϕ.

Example 11 *By setting a minimum conjunctive support threshold, we define an anti-monotone constraint, commonly called the frequency constraint. Likewise, the disjunctive frequency constraint, relying on a minimum disjunctive support threshold, is a monotone one.*

Hereafter, $Supp(\wedge I)$ will simply be denoted $Supp(I)$. In addition, if there is no risk of confusion, *conjunctivesupport* will be called *support*. The next section will focus on frequent itemsets, induced by the frequency constraint.

2.2.3 Frequent Itemsets

Since in practice, in the first part of this thesis, we are mainly interested in itemsets that occur at least in a given number of transactions, we introduce the notion of $frequency$.

Definition 8 *(Frequency of an itemset)*
The frequency of an itemset $I \subseteq \mathcal{I}$ in a context $\mathcal{D}$, denoted by Freq(I), is equal to $Freq(I) = \frac{Supp(I)}{|\mathcal{O}|}$

Example 12 *The frequency of the itemset ACE from Figure 2.1 is $\frac{Supp(ACE)}{|\mathcal{O}|} = \frac{2}{7} =$ 0.28571*

In the remainder, we will mainly use the support of itemsets instead of their frequency.

Definition 9 *(Frequent or Infrequent itemset)*
An itemset I is said to be frequent in $\mathcal{D}$ if Supp(I) is greater than or equal to a user-specified threshold, denoted minsupp. Otherwise, I is said to be infrequent or rare.

Example 13 *Consider the itemset BC from the context given in Figure 2.1. All transactions 2, 3, 6 and 7 contain the itemset. Therefore, Supp(BC) = 4. The frequency of BC is then equal to $\frac{4}{7}$. If minsupp = 2, then BC is considered as frequent in $\mathcal{D}$ since Supp(BC) = 4 $\geq$ 2.*

In the remainder, as well as tables and figures presenting experimental results, the value of *minsupp* will be denoted by θ. By setting the *minsupp* threshold, we only consider frequent itemsets (and not the hole set of itemsets). Hereafter, we will denote by $\mathcal{FI}$ the set of frequent itemsets that can be extracted from a context $\mathcal{D}$ for a given *minsupp*. The next proposition sheds light on an important property of the set of frequent itemsets. It states that all subsets of a frequent itemset are also frequent. Conversely, the supersets of an infrequent itemset are also infrequent.

Proposition 2 *Let $I \subseteq \mathcal{I}$. We have [49]:*

- *If $I \in \mathcal{FI}$, then $\forall I_1 \subseteq I$, $I_1 \in \mathcal{FI}$.*
- *If $I \notin \mathcal{FI}$, then $\forall I_1 \supseteq I$, $I_1 \notin \mathcal{FI}$*

This result comes from the fact that the constraint induced by setting minsupp is anti-monotone (*cf.* Definition 6). Since the supersets of infrequent itemsets are expected to be infrequent, set $\mathcal{I}$ (and consequently context $\mathcal{D}$) will be reduced to frequent items. Infrequent ones will thus be pruned. The set of frequent itemsets induces an order ideal (or down-set) in $(\mathcal{P}(\mathcal{I}), \subseteq)$ when partially ordered *w.r.t.* set inclusion. An ideal order is defined in the following.

Definition 10 *(*Ideal Order*)*
A subset $\mathcal{S}$ of $\mathcal{P}(\mathcal{I})$ is an ideal order in $(\mathcal{P}(\mathcal{I}), \subseteq)$ if it fulfills the following properties [50]:

- *If $x \in \mathcal{S}$, then $\forall y \subseteq x, y \in \mathcal{S}$*
- *If $x \notin \mathcal{S}$, then $\forall y \supseteq x, y \notin \mathcal{S}$*

Set $\mathcal{S}$ is hence downwardly closed since for each $x \in \mathcal{S}$, all its subsets are in $\mathcal{S}$.

An ideal order splits the power-set of items into two disjoint parts: The first contains itemsets fulfilling the associated constraint (i.e. $frequency constraint$), while the second part contains those not fulfilling it. Both parts are delimited thanks to a positive border and a negative one, respectively [51]. The positive border contains the $maximal elements, w.r.t.$ set inclusion, among those that fulfill the constraint associated to the ideal order. While the negative border gathers the $minimal elements, w.r.t.$ set inclusion, among those that do not fulfill the constraint. These borders are formally defined as follows:

Definition 11 *(*Positive, Negative border*)*
Let $(\mathcal{P}(\mathcal{I}), \subseteq)$ be a partially ordered set of elements and $\mathcal{S}$ be a subset of $\mathcal{P}(\mathcal{I})$ s.t. $\mathcal{S}$ is an order ideal in $(\mathcal{P}(\mathcal{I}), \subseteq)$. $\mathcal{S}$ can be represented by its positive border $\mathcal{B}_d^+(S)$ or its negative border $\mathcal{B}_d^-(S)$ defined as follows:

$$\mathcal{B}_d^+(S) = max_{\subseteq}\{I \in \mathcal{S}\}$$
$$\mathcal{B}_d^-(S) = min_{\subseteq}\{I \in \mathcal{P}(\mathcal{I}) \backslash \mathcal{S}\}$$

Dually, a monotone constraint induces an **order filter** [50] in $(\mathcal{P}(\mathcal{I}), \subseteq)$. If an element belongs to this latter order, then it is the same for all its supersets (*cf.* Definition 7).

2.2.4 Concise Representations for Frequent Itemsets

Several reported works shed light on the huge number of frequent itemsets extracted from a given context. In this situation, extracting a subset of itemsets constitutes an interesting solution for concisely representing frequent itemsets [44, 45, 52, 53]. To be lossless, this subset should enable the derivation of the whole set of frequent itemsets, associated to their exact supports. In this case, it is called *exact representation of frequent itemsets.* Definition 12 summarizes this concept:

Definition 12 *(*Exact concise representation of frequent itemsets*)*
Let ε be a set of itemsets. ε is said to be an exact concise representation of the set of frequent itemsets if, starting from ε, we are able to guess whether an arbitrary itemset I is frequent or not. In addition, if I is frequent, then we can exactly determine its conjunctive support.

In fact, the concept of concise representation for frequent itemsets is derived from a more general framework, called the ϵ-adequate representation introduced in [54]. We begin by describing this framework, then we adapt it to our context. Intuitively, an ϵ-adequate representation is a representation which can substitute another one in order to answer the same request(s), more effectively, possibly at the cost of an error bounded by parameter ϵ. Such a representation is defined as follows:

Definition 13 *(*ϵ-adequate representation*)*
Let $\mathcal{S}$ be a class of structures. Let $\mathcal{Q}$ be a class of queries for $\mathcal{S}$. The value of a query $Q \in \mathcal{Q}$ on a structure $s \in \mathcal{S}$ is assumed to be a real number in [0, 1] and is denoted by $Q(s)$. An ϵ-adequate representation for $\mathcal{S}$, w.r.t. a class of queries $\mathcal{Q}$, is a class of structures $\mathcal{C}$, a representation mapping $rep : \mathcal{S} \mapsto \mathcal{C}$ and a query evaluation function $m : \mathcal{Q} \times \mathcal{C} \mapsto [0, 1]$ s.t. $\forall Q \in \mathcal{Q}, \forall s \in \mathcal{S}, \mid Q(s) - m(Q, rep(s)) \mid \leq \epsilon$.

In our case, the class of structures $\mathcal{S}$ is composed by the different sets of frequent itemsets that can be drawn from all possible binary extraction contexts $\mathcal{EC}$, defined over a set of items $\mathcal{I}$, a set of objects $\mathcal{O}$, and for a minimum support threshold θ. Thus, we have $\mathcal{S} = \{\mathcal{FI}_{\mathcal{D}} \mid \mathcal{D} \in \mathcal{EC}\}$. The set of queries represents those searching for the frequency of itemsets of no more than $\mid \mathcal{I} \mid$ size. This set is as follows: $\mathcal{Q} = \{Q_X \mid X \subseteq \mathcal{I}\}$ where the value of Q_X in a context $\mathcal{D} \in \mathcal{EC}$ is defined by $Q_X(\mathcal{D}) = Freq(X) = \frac{Supp(X)}{|\mathcal{O}|}$. While *rep* is a given concise representation of frequent itemset, $\mathcal{C}$ is the application of *rep* on the different contexts of $\mathcal{EC} : \mathcal{C} = \{rep(\mathcal{D} \mid \mathcal{D} \in \mathcal{EC})\}$. Finally, m is the function by which the frequency of an arbitrary itemset is assessed starting from the *rep* representation.

To establish the link between an exact concise representation of frequent itemsets and the concept of ϵ-adequate-representation, we note that exact representations form a 0-adequate-representation of the set of frequent itemsets. Indeed, for an arbitrary context and a given value θ minimum support, they allow the exact retrieval of the respective

frequencies of frequent itemsets. Error ϵ is hence equal to 0. This is not the case of approximate concise representations, like the δ-free set-based one and maximal frequent itemsets [55], from which only an approximation is possible when searching for the frequency of an arbitrary itemset.

An exact concise representation is also called *perfect cover* if it fulfills the conditions stated by the following definition:

Definition 14 *(Perfect cover)*
A cover of a set of patterns $\mathcal{S}$ is a set $\mathcal{S}_1$ that allows recovering $\mathcal{S}$ without information loss. $\mathcal{S}_1$ will be said perfect if it is always a subset of $\mathcal{S}$.

It is also important to note that exact concise representations are preferable to the whole set of frequent itemsets *w.r.t.* the minimal description length principle [56]. This principle states that the best theory describing a set of data is the one minimizing the description length of the theory plus the description length of the data described (or compressed) by the theory. It seeks to minimize the description length of the entire data. In the general case, this principle can be roughly described as follows:

Definition 15 *(Minimum description length principle)[56]*
Given a set of hypotheses $\mathcal{H}$ learned from a set of data D, the best hypothesis $H \in \mathcal{H}$ is the one that minimizes:

$$L(D, H) = L(H) + L(D \mid H)$$

in which

- *$L(H)$ is the length in bits of the description of H, and,*
- *$L(D \mid H)$ is the length, in bits, of the description of data D when encoded with H.*

If we bring this principle into the context of concise exact representations of frequent itemsets, then the description length of the theory (here, a concise representation $\mathcal{CR}$) given the input data (here, the set of $\mathcal{FI}_{\mathcal{D}}$ of frequent itemsets associated to a context $\mathcal{D} \in \mathcal{EC}$) is computed as: $\mathrm{L}(\mathcal{FI}_{\mathcal{D}}, \mathcal{CR}) = \mathrm{L}(\mathcal{CR}) + \mathrm{L}(\mathcal{FI}_{\mathcal{D}} \mid \mathcal{CR})$. As a consequence, the minimum description length principle thus seeks a concise representation that minimizes $\mathrm{L}(\mathcal{FI}_{\mathcal{D}}, \mathcal{CR})$.

To reduce the size of pattern sets, different proposals rely on FCA [50]. **Closed frequent itemsets** ($\mathcal{CFI}$) [7] grasped a growing interest in the last decade. The next section is dedicated to its mathematical background issued from FCA.

2.3 Formal Concept Analysis

FCA mathematical foundations [50] have been used as a theoretical basis for various tasks [57]. In our context, $\mathcal{CFI}$ as a concise representation of frequent patterns introduced in [7] is based on large proposals issued from FCA. Let us recall its basic constructs.

2.3.1 Galois Connection

We begin by defining the Galois connection used to make the link between the power-sets $\mathcal{P}(\mathcal{I})$ and $\mathcal{P}(\mathcal{O})$ associated respectively to the set of items $\mathcal{I}$ and the set of objects $\mathcal{O}$.

Definition 16 *(Galois Connection)*
Let $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{M})$ be an extraction context. The application ψ is defined from the power-set of objects (i.e. $\mathcal{P}(\mathcal{O})$) to the power-set of items (i.e. $\mathcal{P}(\mathcal{I})$). It associates to a set of objects O the set of items $i \in I$ which are common to all objects $o \in O$:

$$\psi : \mathcal{P}(\mathcal{O}) \rightarrow \mathcal{P}(\mathcal{I})$$
$$O \mapsto \psi(O) = \{\ i \in \mathcal{I} \mid \forall\ o \in O, (o,i) \in \mathcal{M}\ \}$$

In a dual way, the application ϕ is defined from the power-set of items (i.e. $\mathcal{P}(\mathcal{I})$) to the power-set of objects (i.e. $\mathcal{P}(\mathcal{O})$). It associates to a set of items I the set of objects $o \in \mathcal{O}$ that contains all items $i \in I$:

$$\phi : \mathcal{P}(\mathcal{I}) \rightarrow \mathcal{P}(\mathcal{O})$$
$$I \mapsto \phi(I) = \{\ o \in \mathcal{O} \mid \forall\ i \in I, (o,i) \in \mathcal{M}\ \}$$

The couple of applications (ψ, ϕ) is a Galois connection between the power-set of $\mathcal{O}$ and that of $\mathcal{I}$ [50].

Definition 17 describes the properties that must fulfill an operator to be qualified as a closure or a kernel one [50].

Definition 17 *(Closure, Kernel Operator)*
Let ($\mathcal{E}$,$\subseteq$) be a partially ordred set and x, y be two elements of $\mathcal{E}$. An operator h defined from ($\mathcal{E}$,$\subseteq$) is called a closure operator if it is:

1. ***Isotone*** *: $x \subseteq y \Rightarrow h(x) \subseteq h(y)$,*
2. ***Extensive*** *: $x \subseteq h(x)$, and,*
3. ***Idempotent*** *: $h(h(x)) = h(x)$.*

Given the closure operator h applied on the partially ordered set ($\mathcal{E}$,$\subseteq$), an element $x \in \mathcal{E}$ is said to be closed if its image by h is equal to itself; i.e. $h(x) = x$.

If an operator h', defined from ($\mathcal{E}$,$\subseteq$) to ($\mathcal{E}$,$\subseteq$), is such that $h'(x) \subseteq x$, then h' has the property of being contractive. If it is also isotone and idempotent, then h' is said to be a kernel operator.

The following definition introduces the closure operators associated to a Galois connection.

Definition 18 *(Galois closure operators)*
Let us consider the power-sets $\mathcal{P}(\mathcal{I})$ and $\mathcal{P}(\mathcal{O})$, with the inclusion relation $\subseteq$, i.e. the partially ordered sets $(\mathcal{P}(\mathcal{I}), \subseteq)$ and $(\mathcal{P}(\mathcal{O}), \subseteq)$. The operators $\gamma = \phi \circ \psi$ from $(\mathcal{P}(\mathcal{I}), \subseteq)$ to $(\mathcal{P}(\mathcal{I}), \subseteq)$ and $\omega = \psi \circ \phi$ from $(\mathcal{P}(\mathcal{O}), \subseteq)$ to $(\mathcal{P}(\mathcal{O}), \subseteq)$ are closure operators of the Galois connection [58]. They define closure systems on $\mathcal{P}(\mathcal{I})$ and $\mathcal{P}(\mathcal{O})$, respectively. The operator γ generates closed subsets of items, while ω generates closed subsets of objects (transactions).

This leads us to the definition of a formal concept.

Definition 19 *(Formal concept)*
A pair $c = (O, I) \in O \times I$ of mutually corresponding subsets, i.e. $O = \psi(I)$ and $I = \phi(O)$, is called a formal concept, where O is called extent of c and I is called its intent.

Example 14 *The pair (36, ABCE) is a concept from the extraction context given by Figure 2.1.*

Having a Galois connection from a given extraction context, the following properties hold for every $I, I_1, I_2 \subseteq \mathcal{I}$ and $O, O_1, O_2 \subseteq \mathcal{O}$ [50]:

(1) $O_1 \subseteq O_2 \Rightarrow \psi(O_2) \subseteq \psi(O_1)$	(1') $I_1 \subseteq I_2 \Rightarrow \phi(I_2) \subseteq \phi(I_1)$
(2) $I \subseteq h_c(I)$	(2') $O \subseteq h'_c(O)$
(3) $I_1 \subseteq I_2 \Rightarrow h_c(I_1) \subseteq h_c(I_2)$	(3') $O_1 \subseteq O_2 \Rightarrow h'_c(O_1) \subseteq h'_c(O_2)$
(4) $h_c(\phi(I)) = \phi(I)$	(4') $h'_c(\psi(O)) = \psi(O)$
(5) $h_c(h_c(I)) = h_c(I)$	(5') $h'_c(h'_c(O)) = h'_c(O)$
(6) $O \subseteq g(I) \Leftrightarrow I \subseteq f(O)$	

2.3.2 Equivalence Classes and Closed Itemsets

Once applied, the closure operator γ induces an equivalence relation on the power-set of items $\mathcal{P}(\mathcal{I})$ splitting it into so-called *equivalence classes* [18], which will further be denoted γ-equivalence classes. A γ-equivalence class is then defined as follows:

Definition 20 *(γ-Equivalence class)*
A γ-equivalence class contains a set of itemsets sharing the same set of objects, hence, having the same closure computed using the operator γ.

Example 15 *Consider the context given by Figure 2.1. Since the itemsets ABC and AE share the same set of transactions, namely {3, 6}, they belong to the same γ-equivalence class. As a result, they have a common closure, namely ABCE.*

In each γ-equivalence class, the largest itemset (*w.r.t.* set inclusion) is called a *closed itemset*. The definition of this particular itemset is given in the following.

Definition 21 *(Closed itemset)*
An itemset $I \subseteq \mathcal{I}$ is said to be closed if $\gamma(I) = I$. [7]

Example 16 *Given the context sketched by Figure 2.1, itemset ABCE is a closed one since it is the maximal set of items common to the set of objects {3, 6}. Itemset AE is not closed since all objects containing itemset AE also contain items B and E.*

Definition 22 *(Set of closed itemsets)*
Consider context $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{M})$ and the closure operator of Galois connection γ. Set $\mathcal{CI}$ of closed itemsets in context $\mathcal{D}$ is defined as follows:

$$\mathcal{CI} = \{I \subseteq \mathcal{I} \mid I \neq \emptyset \wedge \gamma(I) = I\}$$

Therefore, a $\mathcal{CFI}$ is defined as follows:

Definition 23 *(Closed Frequent Itemset $\mathcal{CFI}$)*
A closed itemset is said to be frequent in context $\mathcal{D}$ if Supp(L) is greater than or equal to a user specified threshold (i.e. θ).

In the remainder of this thesis, we will denote by $\mathcal{CFI}$ the set of closed frequent itemsets that can be extracted from a context $\mathcal{D}$ for a given θ.

As stated by Proposition 2, the same properties hold over the set of $\mathcal{CFI}$.

Proposition 3 *Let $I \subseteq \mathcal{I}$. We have [49]:*
- *If $I \in \mathcal{CFI}$, then $\forall I_1 \subseteq I$, $I_1 \in \mathcal{CFI}$,*
- *If $I \notin \mathcal{CFI}$, then $\forall I_1 \subseteq I$, $I_1 \notin \mathcal{CFI}$, and,*
- *if $I \in \mathcal{FI}$, then Supp(I) = Supp(γ(I)).*

2.4 Elements of Information Theory

While formal concepts have defined the environment for mining interesting correlation between data in a dataset, based on the frequency measure (i.e. θ), Information theory ($\mathcal{IT}$) has allowed researchers to introduce a new measure, namely the joint entropy, from which, a new set of itemsets has been defined. Indeed, the problem of informative itemsets mining has been widely studied in the last few years. It has allowed to identify more hidden correlations from binary datasets that provide a meaningful and additional distinction of items. In other words, the task of mining informative itemsets sums up in selecting a small set of binary items (features) which provide a distinction -as good as possible- of the hole set of objects (transactions, documents, etc.) in a given dataset.

In what follows, we give most important definitions that will be of use in the remainder of this thesis. More details are sketched in chapter 3 section 3.2.3 page 67.

Based on information theoretical notions presented in [6], we begin by defining the notion of joint entropy of an itemset, a measure for the amount of information conveyed by itemsets.

Features (items)	Documents (obejcts)						
	d_1	d_2	d_3	d_4	d_5	d_6	d_7
A	0	0	1	0	1	1	0
B	0	1	1	1	1	1	1
C	1	1	1	0	0	1	1
D	1	0	0	0	1	0	1
E	1	1	1	1	0	1	1

Figure 2.2: Features in documents

Definition 24 *(Joint Entropy)*
Let $I = i_1, i_2, \ldots, i_k \in \mathcal{I}$ an itemset and $B = b_1, b_2, \ldots, b_k \in \{0,1\}^k$ be a tuple of binary values. The joint entropy of I is defined as follows:

$$H(I) = -\sum_{B \in \{0,1\}^k} P(i_1 = b_1, \ldots, i_k = b_k) log P(i_1 = b_1, \ldots, i_k = b_k)$$

Example 17 *Let Figure 2.2 be a binary dataset (a binary projection issued from Figure 2.1). The joint entropy of DE is given by $H(DE) = -\frac{3}{7}log(\frac{3}{7}) - \frac{3}{7}log(\frac{3}{7}) - \frac{1}{7}log(\frac{1}{7}) = 0.436$ where quantities $\frac{3}{7}$ and $\frac{3}{7}$ and $\frac{1}{7}$ respectively represent the joint probabilities of projection values $(1,1)$ and $(0,1)$ and $(0,0)$ in the database.*

Therefore, we define a *miki* as follows:

Definition 25 *(Maximally Informative k-Itemsets)*
An itemset $I \in \mathcal{I}$ of cardianlity k is a miki, if $\forall J \in \mathcal{I}$ of cardianlity k

$$H(J) \leq H(I)$$

Note that the joint entropy of an itemset increases as more items are added to it. Because items are binary features, every item provides at most 1 bit of additional information.

To prune the search space of itemsets, the monotonicity constraint over the set of *miki*s holds.

Proposition 4 *Let $X, Y \in \mathcal{I}$ two itemsets such that $X \subseteq Y$. Then*

$$H(X) \leq H(Y)$$

Proposition 4 shows that joint entropy is a non-decreasing function of the number of items involved in the itemset. This raises the issue of choosing a good value for parameter k. In theory, larger values of k will give a better distinction between transactions. On the other hand, feature selection calls for small numbers of items. In many cases, the right

value of k will be implied by the application. The problem is very similar to the selection of the right number of clusters in clustering tasks, where this number is often determined by considering increasing values and stopping when there is a clear drop in improvement.

More details are illustrated in the next chapter to deal with the problem of *miki* mining from big data as well as over a real time constraint of dynamic data.

2.5 Conclusion

In this chapter, we have given an overview of basic definitions that will be useful in the remainder of this thesis. Firstly, we have sketched the mathematical background of FCA as a solution to reduce the huge number of frequent itemsets that can be extracted from a dataset. Secondly, we have presented two important definitions introducing the problem of mining *miki* from a given extraction context.

In the next chapter, we will give a throughout survey of the main algorithms for mining $\mathcal{CFI}$s and *miki* proposed in the literature.

Chapter 3

State of the Art

3.1 Introduction

In this chapter, we present the general concept of knowledge discovery $\mathcal{KD}$ [59]. In particular, we put the focus on the problem of the Closed Frequent Itemset ($\mathcal{CFI}$) mining and we discuss the main recent existing techniques and methods that have been proposed to solve them. In addition, we address the problem of mining maximally informative k-itemsets (*miki*) and we discuss the different approaches and techniques that have been proposed to handle them.

Second, we investigate and detail the different working processes of the recent, existing parallel and distributed mining algorithms. In particular, we address the problem of parallel mining of $\mathcal{CFI}$s and *miki* in massively distributed environments.

3.2 Knowledge Discovery

Knowledge discovery [59] is the whole process of identifying new, potentially useful patterns in data. Data mining presents a core step of a knowledge discovery process. It wraps a set of techniques and methods that allow extracting new knowledge from data.

In this section, we discuss the state of the art of these different techniques. Especially, we present the problem of mining $\mathcal{CFI}$ and *miki* and we discuss the main methods and techniques that have been proposed in the literature to solve them.

3.2.1 Main Frequent-Itemsets Mining Algorithms

As illustrated in the previous chapter, the problem of Frequent-Itemsets mining ($\mathcal{FIM}$) was first introduced in [5]. A naive approach to determine all frequent itemsets in dataset $\mathcal{D}$ simply consists in determining the *support* of all item combinations in $\mathcal{D}$. Then only the items/itemsets that satisfy a given minimum support θ are retained. However, this approach is very expensive, since it results in much of I/O disc access (*i.e.* dataset scans).

In the literature, there have been various proposed algorithms to solve the problem of $\mathcal{FIM}$ [46, 16, 17], etc. Despite their different logic and working principles, the main purpose of these algorithms is to extract all frequent itemsets from dataset $\mathcal{D}$ with a minimum support θ specified as a parameter. In the following, we discuss the basic main $\mathcal{FIM}$ algorithms that have been put forward in the literature, namely Apriori and FP-Growth algorithms.

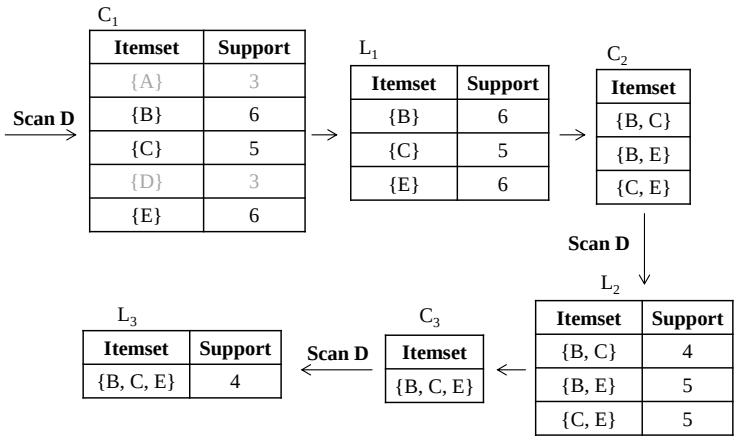


Figure 3.1: Frequent itemset mining using Apriori

Apriori Algorithm: The Apriori algorithm was first introduced in [5]. Its main motivation was to reduce the I/O disc access when mining frequent itemsets. To this end, the Apriori algorithm relies on an anti-monotonicity criterion; *i.e.* if an item/itemset is not frequent, then all of its super-sets cannot be frequent. To extract the frequent itemsets, Apriori scans dataset $\mathcal{D}$ and determines a candidate list C_1 of the frequent items of size one. After that, the algorithm filters C_1 and keeps only the items that satisfy the minimum support and stores them in a list L_1. From L_1, the algorithm generates candidate itemsets of size two in a list, say C_2, by combining all pairs of frequent items of size one in L_1. Subsequently, Apriori scans $\mathcal{D}$ and determines all itemsets in C_2 that satisfy the minimum support. The result is stored in a list L_2. The mining process of Apriori is carried out until there is no more candidate itemsets in $\mathcal{D}$ to be checked.

Despite the anti-monotonicity property, Apriori has shown several drawbacks. This is particularly the case when the minimum support is very low (which implies a huge number of frequent itemset candidates). In this case, the Apriori algorithm would perform

multiple dataset scans until determining the count of the most lengthy itemset in dataset. This behavior of the APRIORI algorithm would result in a poor performance. The performance of the APRIORI algorithm is proportional to the number of itemset candidates to be checked against dataset.

Example 18 *Figure 3.1 shows a working example of the* APRIORI *algorithm over dataset $\mathcal{D}$ of Figure 2.1. In this example, an itemset is frequent, if it occurs at least 4 times in $\mathcal{D}$. After the first dataset scan, we have three frequent items (BCD) in L_1. An itemset candidate generation step is carried out to build the candidate itemsets of size two (BC, BE, CE) in C_2. From C_2 a list of frequent itemsets of size two, L_2, is returned by filtering the itemset candidates in C_2 based on their support count. Next, from list L_2, an itemset candidate step generation is executed to determine the candidate itemsets of size three (BCE) in C_3. Finally, a last dataset scan is performed to filter C_3 and keeps only the frequent itemsets of size three in L_3. Since there is no more candidate itemsets, the algorithm stops.*

In example 18, with a support of 4, APRIORI performs 3 dataset scans in total to determine all frequent itemsets of size 3. With a low minimum support, there will be more frequent itemsets in $\mathcal{D}$, which implies more dataset scans. For example, if there are 10^4 frequent items, then the APRIORI algorithm will need to generate more than 10^7 candidate itemsets of size 2 and the supports of all of these candidates will be checked in $\mathcal{D}$. Thus, the overall performance of the APRIORI algorithm is highly impacted when the minimum support is low (*i.e.* a high number of candidate itemsets).

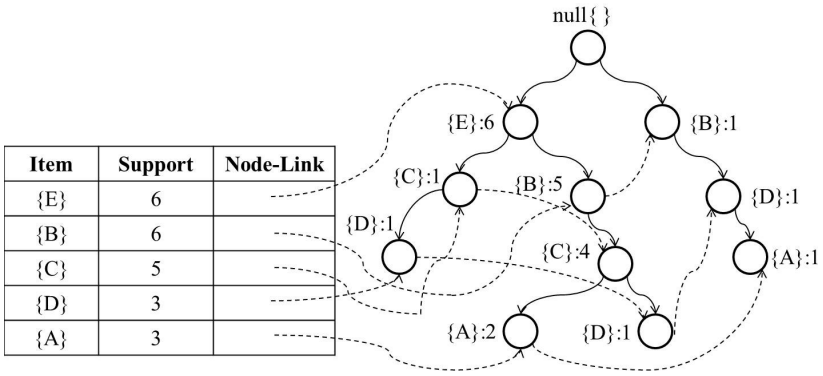


Figure 3.2: Frequent itemset mining using FP-Growth

FP-Growth Algorithm: FP-GROWTH (Frequent-Pattern Growth) algorithm [46] has been considered as the most powerful technique in $\mathcal{FIM}$. The popularity that the FP-GROWTH algorithm has gained is related to its none-candidate generation feature. Unlike previously mentioned techniques, the FP-GROWTH algorithm does not rely on any itemset candidate generation approach. To determine the frequent itemsets, FP-GROWTH

accesses a dataset twice *i.e.* to filter out the none-frequent items and compress the whole dataset in an Fraquent-Pattern tree (FP-tree) structure. Once the FP-tree is built, the algorithm uses a recursive divide-and-conquer approach to mine the frequent itemsets from the FP-tree. Thus, the FP-GROWTH algorithm performs two dataset scans. The following describes each pass over dataset.

Pass 1: Similar to theAPRIORI algorithm, the FP-GROWTH's first pass over $\mathcal{D}$ is devoted to determine the *support* count of each item in $\mathcal{D}$. The algorithm retains only the frequent items in list L. Afterwards, FP-GROWTH sorts L in a descending order according to the *support* counts of the items.

Pass 2: The algorithm creates the root of the tree, labeled with "null", then scans dataset $\mathcal{D}$. The items in each transaction are processed in L order (*i.e.,* sorted according to descending *support* count), and a branch is created for each transaction. Each node in the FP-tree accounts for an item in L, and each node is associated with a counter (*i.e.* *support* count initialized to 1). If a transaction shares a common *prefix* with another transaction, then the *support* count of each visited node is incremented by 1. To facilitate the FP-tree traversal, an item header table is built so that each item will point to its occurrences in the tree via a chain of node links.

Mining FP-tree: The FP-tree is mined as follows. Start from each frequent item (*i.e.* pattern) of size 1 (as an initial suffix pattern), construct its conditional pattern base (a *sub-dataset*, which consists of the set of prefix paths in the FP-tree co-occuring with the suffix pattern), then construct its (conditional) FP-tree, and perform the mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Example 19 *Let us consider dataset $\mathcal{D}$ shown in Figure 2.1. In this example, we consider an itemset to be frequent if it occurs at least twice in $\mathcal{D}$. After the first pass over dataset $\mathcal{D}$,* FP-GROWTH *returns list L of frequent items. In this example, we have $L = \{\{A:3\},\{B:6\},\{C:5\},\{D:3\},\{E:6\}\}$. Next, it sorts list L of frequent items in a descending order according to their support counts. Hence, L becomes $L = \{\{E:6\},\{B:6\},\{C:5\},\{D:3\},\{A:3\}\}$. By scanning $\mathcal{D}$, an FP-tree is built according to the order of items in L. The left part of Figure 3.2 shows a header table that contains the information about each node in the constructed FP-tree (right part of Figure 3.2) of our example. To mine the constructed FP-tree, we consider the last item in L, A. Item A occurs in two FP-tree branches (The right side of Figure 3.2). The occurrences of A can easily be found by its chain of node links. The paths formed by these branches are $<$ EBCA: 2 $>$ and $<$ BDA: 1 $>$. Considering item A as a suffix, its corresponding two prefix paths are $<$ EBC: 2 $>$ and $<$ BD: 1 $>$ which form its conditional pattern base. Using this conditional pattern base as a transaction dataset,* FP-GROWTH *builds an A-conditional FP-tree which contains only a single path $<$ EBC: 2 $>$. Here, item D is not included because its support count is* 1 *(not frequent). The single path $<$ EBC: 2 $>$ generates all frequent itemsets that involve item A (ABCE: 2, ABC: 2, ABE: 2, ACE: 2, AB: 3, AC: 2, AE: 2). Likewise, item D*

occurs in 3 *FP-tree branches. The paths formed by these* 3 *branches are < ECD: 1 >, < EBCD: 1 > and < BD: 1 >. Considering item D as a suffix, then its corresponding prefix paths are < EC: 1 >, < EBC: 1 > and < B: 1 > which form its conditional pattern base.* FP-Growth *builds a D-conditional FP-tree which contains two paths < EC: 2 > and <B: 2 >. From these two paths,* FP-Growth *generates all frequent itemsets involving item D (CDE: 2, BD: 2). Similar for item C, its C-conditional FP-tree contains two paths: < E: 5 > and <B: 4 >.* FP-Growth *generates the frequent itemsets involving item C, (CE: 5, BC: 4). For item B, its B-conditional FP-tree contains a single path < E: 5 >. Thus the frequent itemset involving B is (BE: 5). The item E does not have any prefix, so the algorithm stops.*

At this point, we notice the very high number of frequent itemsets, which will disrupt their exploitation and render their interpretation, by human experts, almost impossible. Hence, the reduction in this large number of frequent itemsets has become of paramount importance for a better handling of the information extracted from datasets.

Some work has been designed to define sets of patterns being as compact as possible. Such a set is called "an exact concise representation" of frequent itemsets. In the literature, exact concise representations have been defined based on several and different tricks, the first of which was the exploration of frequent closed itemsets. This approach has been proposed in order to avoid several drawbacks; the most remarkable is the reduction of the very large number of the itemsets to mine. This approach is based on the mathematical foundations of the formal concept analysis introduced in [50], and it has allowed remarkably a decrease in the cost of $\mathcal{FIM}$ extraction. In the next section, we detail algorithms dedicated to mine the set of $\mathcal{CFI}$s.

3.2.2 Closed-Frequent-Itemsets-Based Algorithms

As previously mentioned, the big growth of the interest in the $\mathcal{FIM}$ is owed to the usefulness of frequent itemsets in many important fields. In fact, thanks to these itemsets, human experts can obtain pertinent correlations between dataset items. Consequently, they can acquire a deeper understanding thanks to the mined patterns. However, in real-life datasets, performed experiments has shown that the number of frequent itemsets is huge when dataset is dense or the minimum support threshold is set too low [45]. This phenomenon makes the exploitation and the handling of such an amount of extracted knowledge very difficult.

In this section, we tackle the problem of mining $\mathcal{CFI}$ as an exact concise representation of the whole set of frequent itemsets. In general, the criterion used to classify $\mathcal{CFI}$-based algorithms is inherited from the $\mathcal{FIM}$ stuff, i.e. the technique used to traverse the search space. Hence, $\mathcal{CFI}$-based algorithms can be roughly split into three categories, namely "test-and-generate", "divide-and-conquer" and "hybrid" [60].

In the first category, adopting the "test-and-generate" technique, the most known algorithms are Close [61], A-Close [7] and Titanic [62]. These algorithms stress on the

optimization of a level-wise process for the discovery of both aforementioned closure systems. As a starting point, they consider the already known meet-irreducible set, i.e. items of the ground set $\mathcal{I}$. This set is further extended by self-joined compound elements using the combinatorial phase of Apriori-Gen [63] during the exploration process. Candidate sets are pruned using the conjunction of statistical metrics (e.g. the support measure θ) and heuristics essentially based on structural properties of closed itemsets.

In the second category, the most known algorithm which adopts the "divide-and-conquer" technique is Closet [64]. The latter introduced the use of the highly compact data structure FP-tree (Frequent-Pattern tree) [46] within the $\mathcal{CFI}$ discovery process. Using a depth-first traversal of the search space, Closet tries to split the extraction context, stored in a global FP-tree, into smaller sub-contexts and to recursively apply the $\mathcal{CFI}$ mining process on these sub-contexts. The mining process also heavily relies on the search space pruning. This pruning is also based on statistical metrics in conjunction with introduced heuristics. Some improvements or alternatives of this algorithm have been proposed, mainly Closet+ [65] and FP-Close [66], while respecting its driving idea.

Algorithms of the third category, adopting the "hybrid" technique, use properties of both previously mentioned techniques. The CHARM [67] algorithm is the most known. Unlike other methods which exploit only the (closed) itemset search space, CHARM simultaneously explores both the search space of closed itemsets and that of transactions thanks to an introduced data structure called the Itemset-Tidset tree (IT-tree) [67]. Each node in the IT-tree contains a $\mathcal{CFI}$ candidate and a list of the transactions to which it belongs. This list is called *tidset* [67]. CHARM explores the search space in a depth-first manner, like the algorithms of the "divide-and-conquer" technique, without splitting the extraction context into smaller sub-contexts. However, it generates each time a single candidate, like the algorithms of the "test-and-generate" technique. Then, it tries to test whether it is a $\mathcal{CFI}$ or not, using *tidset* intersections and subsumption checking. The mining process also heavily draws on the search space pruning. This pruning is based on imposed statistical metrics in conjunction with introduced heuristics.

Two other algorithms are part of the third category, namely Dci-Closed [68] and LCM [69]. Both algorithms can only be considered as improvements of CHARM as they inherit the use of tidsets and the hybrid traverse of the search space adopted in CHARM. Nevertheless, they avoid the main drawback of previously proposed algorithms, namely the high cost of subsumption checking which allows discarding candidates whose closures have been already mined (case of the second and third category). To this end, Dci-Closed and LCM traverse the search space in a depth-first manner. After discovering a $\mathcal{CFI}$ (i.e. I), they generate a new itemset candidate by extending the $\mathcal{CFI}$ with a frequent item i, $i \in I$. Hence, both algorithms extract the set of $\mathcal{CFI}$s in a linear time of its size [70, 71]. In addition, these algorithms do not need to store, in the main memory, the set containing the previously mined $\mathcal{CFI}$s since they do not require performing subsumption checking. The differences between Dci-Closed and LCM are the strategies for taking closure and the adopted data structures for storing the extraction context in the main memory.

Consequently, we present in what follows some features (or dimensions) which permit highlighting more the major differences between the $\mathcal{CFI}$-based algorithms.

1. **Exploration technique**: Three techniques are used to explore the search space, namely "test-and-generate", "divide-and-conquer" and "hybrid".
2. **Architecture**: The architecture dimension depends on how a given algorithm is designed: a sequential function in a centralized single processor architecture, or more suited to a parallel treatment in a multiprocessor or distributed architecture (e.g., CLOSET).
3. **Parallelism strategy**: Parallel algorithms can be further described as task, data or hybrid parallelism.
4. **Data source type**: This feature indicates the type of input data: (extended) market basket data (also known as horizontal data), vertical data, relational data, plain text, multimedia data, etc.
5. **Information storage**: Different data structures are used to keep track of the information required for an algorithm execution (storage of candidates, the extraction context, etc). The most privileged structure seems to be the trie structure.
6. **Closure computation**: The closure of an itemset I can be computed thanks to the following two different methods. In the first method, the closure of I is the result of the intersection of the transactions to which it belongs. In the second method, its closure is incrementally computed by searching for items that verify the following property issued from the formal concept analysis presented in chapter 2: $\psi(I) \subseteq \psi(i) \Rightarrow i \in \gamma(I)$, such that $i \in \mathcal{I}$ and $i \notin I$. The closure computation can also be performed on-line or off-line.

In this section, we present a structural and analytical comparative study of FCI-based algorithms, towards a theoretical and empirical guidance to choose the most adequate mining algorithm. The main report of this comparative study is to shed light on an "obsessional" algorithmic effort to reduce the computation time of the interesting itemset extraction step. The obtained success is primarily due to an important programming effort combined with strategies for compacting data structures in the main memory. However, it seems obvious that this frenzied activity loses sight of the essential objective of this step, i.e. to extract reliable knowledge of exploitable size for end users. Thus, almost all these algorithms were focused on enumerating $\mathcal{CFI}$s, presenting a frequency of appearance considered to be satisfactory.

3.2.3 Maximally Informative k-Itemsets mining

The co-occurrence frequency of itemsets (or featureset) in a dataset does not give much information about the hidden correlations between the itemsets. For instance, an itemset, say AB, can be frequent, but the items (or features) inside AB can be redundant. Thus, if we know A, we may not need B. Therefore, besides the frequency criterion of itemsets as a measure of informativeness or interestingness, other measures have been proposed such

as the joint entropy. In the following discussion, we address the problem of *miki* mining. First, we introduce the basic definitions and notations of the *miki* problem that will be used in the remainder of this thesis. Second, we discuss the main existing methods and techniques that have been proposed in the literature to extract *miki*s.

Features	Documents									
	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
A	1	1	1	1	1	0	0	0	0	0
B	0	1	0	0	1	1	0	1	0	1
C	1	0	0	1	0	1	1	0	1	0
D	1	0	1	1	1	1	1	1	1	1
E	1	1	1	1	1	1	1	1	1	1

Figure 3.3: Features in documents

The following definitions introduce the basic requirements for mining *miki* [23].

Definition 26 *(Feature entropy)*
The entropy [72] of feature i in dataset $\mathcal{D}$ measures the expected amount of information needed to specify the state of uncertainty or disorder for feature i in $\mathcal{D}$. Let i be a feature in $\mathcal{D}$, and $P(i = n)$ be the probability that i has value n in $\mathcal{D}$ (we consider categorical data, where the value will be '1' if the object has the feature and '0' otherwise). The entropy of feature i is given by:

$$H(i) = -(P(i = 0)log(P(i = 0)) + P(i = 1)log(P(i = 1)))$$

where the logarithm base is 2.

Definition 27 *(Binary projection)*
The binary projection, or projection of itemset X in transaction T ($proj(X, T)$) is the set of size $|X|$ where each item (i.e., feature) of X is replaced by '1' if it occurs in T and by '0' otherwise. The projection counting of X in dataset $\mathcal{D}$ is the set of projections of X in each transaction of $\mathcal{D}$, where each projection is associated with its number of occurrences in $\mathcal{D}$.

Example 20 *Let us consider Figure 3.3. The projection of (B, C, D) in d_1 is $(0, 1, 1)$. The projections of (D, E) on the same dataset are $(1, 1)$ with nine occurrences and $(0, 1)$ with one occurrence.*

Definition 28 Joint entropy
Given itemset $X = \{x_1, x_2, \ldots, x_k\}$ and a tuple of binary values $\mathcal{B} = \{b_1, b_2, \ldots, b_k\} \in \{0\,1\}^k$, the joint entropy of X is defined as:

$$H(X) = -\sum_{\mathcal{B}\in\{0,1\}^{|k|}} J \times log(J)$$

where $J = P(x_1 = b_1, \ldots, x_k = b_k)$ *is the joint probability of* $X = \{x_1, x_2, \ldots, x_k\}$.

Given dataset $\mathcal{D}$, the joint entropy $H(X)$ of itemset X in $\mathcal{D}$ is proportional to its size k; *i.e.* the increase in the size of X implies a rise in its joint entropy $H(X)$. The higher the value of $H(X)$, the more information the itemset X provides in $\mathcal{D}$. For simplicity, we use the term *entropy* of an itemset X to denote its joint entropy.

Example 21 *Let us consider dataset of Figure 3.3. The joint entropy of* (D, E) *is given by* $H(D, E) = -\frac{9}{10}log(\frac{9}{10}) - \frac{1}{10}log(\frac{1}{10}) = 0.468$, *where the quantities* $\frac{9}{10}$ *and* $\frac{1}{10}$ *respectively represent the joint probabilities of projection values* $(1, 1)$ *and* $(0, 1)$ *in dataset.*

Definition 29 *(*Maximally Informative k-iIemset*)*
Given a set $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ *of features, an itemset* $X \subseteq \mathcal{F}$ *of length* k *is a miki, if for all itemsets* $Y \subseteq \mathcal{F}$ *of size* k, $H(Y) \leq H(X)$. *Hence, a miki is the itemset of size* k *with the highest joint entropy value.*

The problem of mining miki presents a variant of itemset mining. It relies on a joint entropy measure for assessing the informativeness brought by an itemset.

Definition 30 *(*Mining Miki*)*
Given dataset $\mathcal{D}$ *which consists of a set of* n *items (features),* $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$. *Given number* k, *the problem of miki mining is to return a subset* $F' \subseteq F$ *with size* k, *i.e.* $|F'| = k$, *having the highest joint entropy in* $\mathcal{D}$, *i.e.* $\forall F'' \subseteq F \wedge |F''| = k, H(F'') \leq H(F')$.

Example 22 *In Figure 3.3, the existence of the attributes* A, B, C, D, E *in the documents* d_1, d_{10} *is represented by a binary tuple (0, 1). Value "1" means that the attribute occurs in the document, and "0" otherwise. We can discern that itemset* DE *is frequent, since items* D *and* E *exist together in nine documents. However, it does not provide help for document retrieval. In other words, given a document* d_x *in our dataset, one might look for the occurrence of itemset* DE *and, depending on whether it occurs or not, it will not be able to decide which document it is. Contrary to itemset* ABC *which is not frequent, as its items rarely appear together in the dataset. And it is troublesome to summarize the value patterns of itemset* ABC. *We could find the corresponding document* O_3 *by providing* ABC *with the values* $< 1, 0, 0 >$. *Similarly, document* O_6 *is found given values* $< 0, 1, 1 >$ *of* ABC. *Even though* ABC *is infrequent, it represents lots of useful information which is hard to summarize. Having the values of each feature composing* ABC, *it becomes much easier to decide exactly which document they belong to. Decidedly* ABC *is a maximally informative itemset of size* $k = 3$.

In the literature of data mining, several endeavors have been made to explore informative itemsets (or featuresets, or sets of attributes) in datasets [5, 46, 73, 23]. Different measures of itemset informativeness (*e.g.* the frequency of itemset co-occurrence in dataset.) have been used to identify and distinguish informative itemsets from non-informative ones. For instance, by considering the co-occurrence of itemsets, several conclusions can be drawn to explain interesting, hidden relationships between different itemsets in data.

In addition to itemsets co-occurrence frequency measure to identify informativeness in the dataset, an efficient technique that was more widespread has been proposed in [74]. The authors in [74] did not only focus on the analysis of the positive implications between the itemset in data (*i.e.,* implications between supported itemsets), but also they take into account the negative implications. To determine the significance of such itemsets implications, they have used a classic statistical chi-squared measure to efficiently figure out the interesting of such itemset rules.

Generally, in the itemset mining problem there is a trade-off between itemset informativeness and pattern explosion (*i.e.* number of itemsets to be computed). Thus, some itemset informativeness measures (*e.g.* the co-occurrence frequency measure with a very low minimum support) would allow for a potential high number of useless patterns (*i.e.* itemsets), and others would highly limit the number of patterns. The authors in [75] put forward an efficient approach that went over regular used itemset informativeness measures, by developing a general framework of statistical models enabling the scoring of the itemsets in order to determine their informativeness. In particular, in [75], the initial focus was on the exponential models to score the itemsets. However, these models are inefficient in terms of execution time, thus, the authors propose to use decomposable models. On the whole, the techniques proposed in [75] and [74] were mainly dedicated to mine in centralized environments, while our techniques are dedicated to parallel data mining in distributed environments.

The authors in [23] utilized an heuristic approach to extract informative itemsets of length k based on maximum joint entropy, such maximally informative itemsets of size k is called *miki*. This approach captures the itemsets that have high joint entropy. An itemset is a *miki* if all of its constructing items shatter data maximally. The items within a *miki* are not excluding, and do not depend on each other. [23] proposes a bunch of algorithms to extract *miki*. A brute force approach consisted of performing an exhaustive search over dataset to determine all *miki* of different sizes. However, this approach is not feasible due to the large number of itemsets to be determined, which results in multiple dataset scans. Another algorithm suggested in [23], namely *ForwardSelection*, consisted in fixing a parameter k that would denote the size of the *miki* to be discovered. In fact, this algorithm proceeds by determining a top n *miki* of size 1 having the highest joint entropy. Afterwords, the algorithm determines the combinations of *1-miki* of size 2 and returns the top n most informative itemsets. The process continues until it returns the top n *miki* of size k. Example 23 illustrates the mining process of this algorithm.

Example 23 *Given the binary dataset $\mathcal{D}'$, as shown in Figure 3.4. $\mathcal{D}'$ contains* 4 *items*

A	B	C	D
1	0	1	0
1	0	1	0
1	0	0	0
1	1	0	0
1	1	1	0
0	0	1	0
0	0	0	0
0	0	0	1

Figure 3.4: Binary dataset $\mathcal{D}'$

and 8 transactions. Suppose that we want to determine miki of size $k = 3$ using the ForwardSelection algorithm.

The algorithm starts by determining the entropy of each item in $\mathcal{D}'$ as follows:

$H(A) = -\frac{5}{8}log(\frac{5}{8}) - \frac{3}{8}log(\frac{3}{8}) = 0.954$
$H(B) = -\frac{6}{8}log(\frac{6}{8}) - \frac{2}{8}log(\frac{2}{8}) = 0.811$
$H(C) = -\frac{1}{2}log(\frac{1}{2}) - \frac{1}{2}log(\frac{1}{2}) = 1$
$H(D) = -\frac{7}{8}log(\frac{7}{8}) - \frac{1}{8}log(\frac{1}{8}) = 0.543$

Item {C} has the highest entropy, thus {C} presents a seed. From this item seed, ForwardSelection generates the miki candidates of size two ({C A}, {C B}, {C D}). A scan to dataset $\mathcal{D}'$ is performed to determine the entropy of each miki candidates of size two. Here, we have:

$H(CA) = -\frac{3}{8}log(\frac{3}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{2}{8}log(\frac{2}{8}) - \frac{2}{8}log(\frac{2}{8}) = 1.905$
$H(CB) = -\frac{3}{8}log(\frac{3}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{3}{8}log(\frac{3}{8}) - \frac{1}{8}log(\frac{1}{8}) = 1.811$
$H(CD) = -\frac{4}{8}log(\frac{4}{8}) - \frac{3}{8}log(\frac{3}{8}) - \frac{1}{8}log(\frac{1}{8}) = 1.405$

Itemset {C A} has the highest entropy, thus the miki of size 2 in $\mathcal{D}'$ is {C A}. For $k = 3$, {C A} presents a seed to construct the miki candidates of size 3 ({C A B}, {C A D}). The same procedure is carried out to determine miki of size 3 by scanning dataset $\mathcal{D}'$ and determining the entropy of each miki candidate ({C A B}, {C A D}).

$H(CAB) = -\frac{2}{8}log(\frac{2}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{2}{8}log(\frac{2}{8}) = 2.5$
$H(CAD) = -\frac{2}{8}log(\frac{2}{8}) - \frac{2}{8}log(\frac{2}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) - \frac{1}{8}log(\frac{1}{8}) = 2.15$

Since itemset {C A B} has the highest entropy, {C A B} is a miki of size 3.

Although the *ForwardSelection* algorithm accounts for a simple and efficient method to determine *miki* of size k, it has major drawbacks. When size k of the itemset to be discovered tends to be very high, there will be a high number of dataset scans, which will impact the overall performance of the algorithm.

The problem of extracting informative itemsets has not only been proposed for mining static datasets. There has been also interesting work in extracting informative itemsets in data streams [76] [77]. The authors in [78] put forward an efficient method for discovering maximally informative itemsets (*i.e.* highly informative itemsets) from data streams based on a sliding window.

Extracting informative itemsets has a prominent role in feature selection [79]. Various techniques and methods have been proposed in the literature to solve the problem of selecting relevant features to be used in classification tasks. These methods fall into two different categories, namely *Filter* and *Wrapper* methods [80]. Filter methods serve to pre-process data before being used for a learning purpose. They aim to determine a small set of relevant features. Yet, these methods only capture the correlations between each feature (*i.e.,* independent variable, attributes or items) and the target class (*i.e.* predictor). They do not take into consideration the inter correlation between the selected features (*i.e.,* if the features are inter correlated, then they are redundant). On the other hand, to determine an optimal set of relevant features, wrapper methods perform a feature's set search that maximizes an objective function (*i.e.* classifier performance). However, these methods need heavy computations (*i.e.* selecting each time a set of features and evaluate an objective function). To solve this problem, *Embedded* [79] methods have been suggested. The main goal is to incorporate the wrapper methods in the learning process.

3.3 Parallel and Distributed Computing

In this section, we introduce the MapReduce programming model and we detail its working principle and its basic architecture.

3.3.1 MapReduce

MapReduce [9] is a parallel framework for large scale data processing. It has gained increasing popularity, as shown by the tremendous success of Hadoop [81], an open-source implementation. Hadoop enables resilient, distributed processing of massive unstructured datasets across commodity computer clusters (*i.e.* set of commodity machines), where each node of the cluster includes its own storage. MapReduce serves two essential functions:

- **map():** The map function is applied to process input data. Generally, the input data is in the form of a file or directory and is stored in the Hadoop File System

(HDFS). The input file is passed to the mapper function line by line. The mapper processes data and creates several small chunks of data.

- **reduce():** The Reducer function is applied to process data that comes from the mapper. After processing, it produces a new output set, which will be stored in the HDFS.

MapReduce Architecture Figure 3.5 illustrates the architecture of MapReduce. The user sends its program to the master node. The latter assigns map and reduce tasks to workers. Data is divided into data splits (*i.e.* in a hadoop file system). Each map worker reads and executes a map task on its data split and writes the results on its disc (local write). After the execution of all map tasks, each map worker sends its results in the form of (key, value) pairs to the reduce workers. Between the map and the reduce phases, a sorting process is carried out where each key is associated with its list of values. Finally, the reducers perform their computing logic and output the final results to an HDFS.

Example 24 *As an example, suppose we are interested in counting the number of times every word appears in a novel. We can split the task among some people, so each takes a page, writes a word on a separate sheet of paper and takes a new page until they finish. This is the map aspect of MapReduce. In addition, if a person leaves, another one takes its place. This exemplifies the MapReduce's fault-tolerant element.*

When all pages are processed, users sort their single-word pages into some boxes, which represent the first letter of each word. Each user takes a box and sorts each word in the

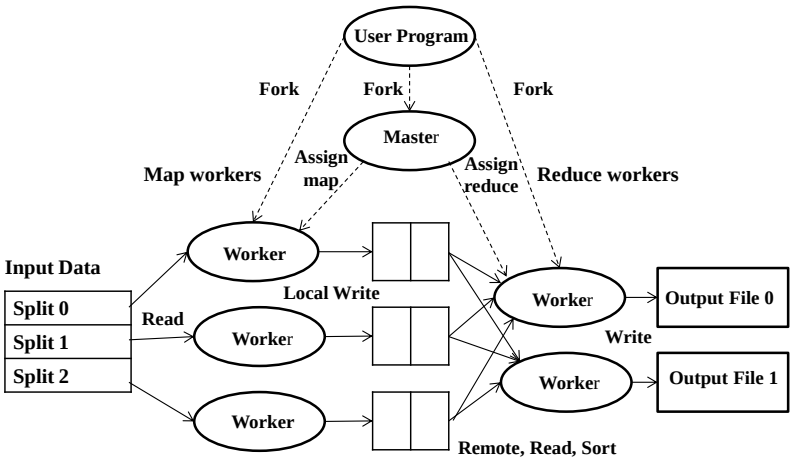


Figure 3.5: MapReduce architecture

stack alphabetically. The number of pages with the same word is an example of the reduce aspect of MapReduce.

3.3.2 Spark and Spark streaming

Apache Spark [10] is an open source cluster computing framework originally developed in the AMPLab at University of California, Berkeley but was later donated to the Apache Software Foundation where it remains today. In contrast to the Hadoop's two-stage disk-based MapReduce paradigm, Spark's multi-stage in-memory primitives provides performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well-suited to machine learning algorithms.

Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos [82]. For distributed storage, Spark can interface with a wide variety, including HDFS and Cassandra [83] or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead. In such a scenario, Spark is run on a single machine with one executor per CPU core. In addition, Spark gives us a comprehensive, unified framework to manage big data processing requirements with a variety of datasets that are diverse in nature (text data, graph data etc.) as well as the source of data (batch vs. real-time streaming data).

When data evolves through time, the latency rate has to be checked fast. Therefore, the Spark streaming framework proposes a fast streaming architecture that allows users to process a very large amount of data through seconds. Indeed, Spark streaming is an essential functionality for the Apache Spark API that provides live data streams that are scalable and fault-tolerant. This helps the Apache Spark engine to more effectively support workloads. It helps to provide good load-balancing, fast failure recovery, integration and other goals and objectives. Unlike traditional systems, tasks are scheduled based on available resources. Dynamic scheduling helps with distribution. These and other design improvements make Spark streaming a popular method for using this Apache open-source tool. Spark streaming is becoming the platform of choice to implement data processing and analytics solutions for real-time data received from Internet of Things (IoT) and sensors. It is used in a variety of use cases and business applications.

Some of the most interesting use cases of Spark streaming include the following [84]:

- Uber, the company behind ride sharing service, uses Spark streaming in their continuous Streaming ETL pipeline to collect terabytes of event data every day from their mobile users for real-time telemetry analytics.
- Pinterest, the company behind the visual bookmarking tool, uses Spark streaming, MemSQL and Apache Kafka technologies to provide insight into how their users are engaging with Pins across the globe in real-time.

- Netflix uses Kafka and Spark streaming to build a real-time online movie recommendation and data monitoring solution that processes billions of events received per day from different data sources.

Other real world examples of Spark streaming include:

- Supply chain analytics
- Real-time security intelligence operations to find threats
- Ad auction platform
- Real-time video analytics to help with personalized, interactive experiences to viewers.

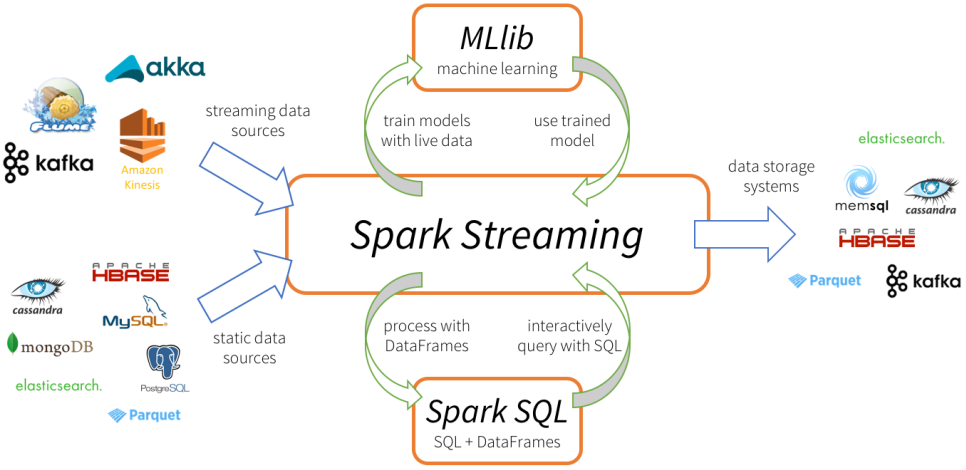


Figure 3.6: Spark streaming framework

Example 25 *Figure 3.7 shows an example of windowed string processing using the Spark streaming framework. To deal with windowed operations, the Spark streaming framework offers the possibility of specifying the size of the sliding window and the sliding interval. In the following example, the size of the sliding window is fixed to three batches of string transactions, and the sliding interval is fixed to one batch.*

To write Spark streaming programs, there are two components we need to know about: DStream and StreamingContext. DStream (short for Discretized Stream) is the basic abstraction in Spark streaming and represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis,

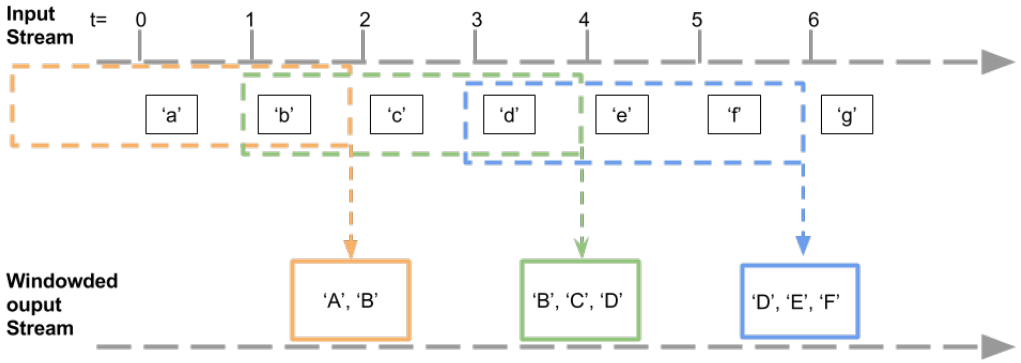


Figure 3.7: Spark streaming example

or by applying operations on other DStreams. Internally, DStream is represented as a sequence of resilient distributed datasets (RDD) objects. Similar to SparkContext in Spark, StreamingContext is the main entry point for all streaming functionalities. StreamingContext has built-in methods for receiving streaming data into a Spark streaming program. Using this context, we can create DStream which represents streaming data from a TCP source, specified as a host name and a port number.

To summarize this section, let us take a look at the various steps involved in a typical Spark streaming program.

- Spark streaming context is used for processing the real-time data streams. Therefore, the first step is to initialize the StreamingContext object using two parameters, SparkContext and sliding interval time. Sliding interval sets the update window in which we process data coming in as streams. Once the context is initialized, no further computations can be defined or added to the existing context. In addition, only one StreamingContext object can be active at the same time.
- After Spark streaming context is defined, we specify the input data sources by creating input DStreams. In our sample application, the input data source is a log message generator that uses Apache Kafka distributed dataset and messaging system. The log generator program creates random log messages to simulate a web server run-time environment where log messages are continuously generated as multiple web applications serve the user traffic.
- Define the computations using the Sparking streaming transformations API like map and reduce to DStreams.
- After streaming computation logic is defined, we can start receiving data and process it using the start method in the StreamingContext object created earlier.
- Finally, we wait for the streaming data processing to be stopped using the awaitTermination method of StreamingContext object.

In the following section, we sketch distributed algorithms for knowledge discovery in data. Some of the algorithms are developed under MapReduce and Spark frameworks.

3.4 Knowledge Discovery and Distributed Computing

With the explosive growth of data, a centralized knowledge discovery process becomes unable to process large volumes of data. The distribution of the computation over several machines has solved the problem; *i.e.* data are located in different commodity machines, and the computation process is distributed over these shared data and executed in parallel.

In the following, first we focus our study on the main parallel $\mathcal{FIM}$ algorithms that have been used in the literature. Second, we sketch parallel approaches for $\mathcal{CFI}$ mining. Finally, we discuss the problem of the parallel discovery of *miki*.

3.4.1 Parallel Frequent-Itemsets Mining

In data mining literature, there have been several endeavours to improve the parallel discovery of frequent itemsets. In the following, we present the major methods and techniques that have been proposed in the literature.

Parallel Apriori Algorithm: In a massively distributed environment, the parallel version of the Apriori (*i.e.* Parallel Apriori) algorithm [85] has shown a better performance than its sequential one. Although the parallelism setting and the availability of high number of resources, the Apriori algorithm has brought regular issues and limitations as shown in its sequential implementation. In a massively distributed environment such as MapReduce, using the Apriori algorithm, the number of jobs required to extract the frequent itemsets is proportional to the size of a lengthy itemset. Hence, with a very small minimum support and a large amount of data, the performance of Parallel Apriori is very poor. This is, because, the inner working process of Apriori algorithm is based on a candidate generation and testing approach which results in a high disc I/O access. In addition, in a massively distributed environment, Apriori algorithm allows for a high data communication between the mappers and the reducers, this is particularly the case when the minimum support tends to be very small.

Parallel FP-Growth Algorithm: The Parallel FP-Growth (PFP-Growth) algorithm [12] has been successfully applied to efficiently extract the frequent itemsets from large datasets. Its high performance in terms of processing time comes as the result of its core mining principle. At its first MapReduce job, PFP-Growth performs a simple counting process to determine a list of frequent items. The second MapReduce job is dedicated to construct an FP-tree to be mined later at the reducer phase. The mining process is carried out in the memory, which explains the high performance runtime of PFP-Growth. Although PFP-Growth has been considered as a highly efficient mining

technique, with a very small minimum support and a large amount of data, it does not scale.

3.4.2 Parallel Closed-Frequent-Itemsets Mining

Many algorithms have been introduced for $\mathcal{CFI}$ mining running in a parallel platform. However, to the best of our knowledge there have been no alternatives at the start of our work in this thesis running under Map/Reduce platform. In the remainder, we will try to depict some of the new alternatives and see the intake in the parallel $\mathcal{CFI}$ mining constraint.

Parallel MT-CLOSED Algorithm: Mining $\mathcal{CFI}$s in a distributed platform poses multiple challenges. In [86], the authors introduced the first parallel shared memory version of the DCI-CLOSED algorithm sketched in section 3.2.2, the MT-CLOSED algorithm. MT-CLOSED algorithm was based on a duplicate checking method that allowed an easy decomposition of the mining tasks into independent sub-tasks with a vertical bitmap representation of dataset. The latter allowed exploiting the nowadays modern processor architectures. Furthermore, the authors were inspired by the cache efficiency of the DCI-CLOSED algorithm and the possibility to take advantage of memory hierarchies in a multi-threaded algorithm.

Parallel AFOPT-CLOSE Algorithm: As in [87], and based on the parallel FP-Growth algorithm PFP [12] which divides an entire mining task into independent parallel subtasks and achieves quasi-linear speedups, AFOPT-CLOSE mines $\mathcal{CFI}$ in four MapReduce jobs and introduces a redundancy filtering approach to deal with the problem of generating redundant itemsets. The approach shows a gain in terms of execution time. However, the number of MapReduce jobs used in the work can be much more reduced and therefor decrease the time execution and gain in terms of effectiveness.

Parallel CLOSET Algorithm: The CLOSET algorithm uses an advanced data structure, based on the notion of sort, called *tree* [11]. The peculiarity of this structure lies in the fact that several transactions will share the same path, of length n in the tree *FP-tree*, if they have the first n items in common. The CLOSET algorithm performs the mining process of closed itemsets in two successive steps [22]. In the first step, it focuses on the construction of *FP-tree*s where the items of size one are ordered by a descending support. For each transaction in the context, the items are processed and a branch is created if needed. In each node of the *FP-tree* structure, there is a counter that keeps track of the number of transactions sharing this node. The second step is dedicated to the exploration of the *FP-trees*. Thus, it begins by considering the 1-itemsets, sorted in ascending order of their respective supports, and examines only their *conditional contexts* (or conditional *FP-trees*) [22]. A conditional context contains only the items that occur with the 1-itemset in question. Therefore, due to lack of parallel algorithms developed

Split	A	B	C	D
S_1	1	0	1	0
	1	0	1	0
	1	0	0	0
	1	1	0	0
	1	1	1	0
S_2	0	0	1	0
	0	0	0	0
	0	0	0	1

Figure 3.8: Data splits of binary dataset $\mathcal{D}'$

under the MapReduce framework, we propose a new version of *Closet*, namely P-Closet, a parallel algorithm designed and implemented using the MapReduce framework. Consequently, after a first pre-processing phase of the transactionnal dataset, each mapper in our architecture will take a conditional sub-context and begin the process of extracting the closed itemsets independently from the other mappers. Before the distribution process of data, we have developed a transactionnal classification approach permetting to establish a split of transactions independent of the other splits.

3.4.3 Parallel Maximally Informative k-Itemsets Mining

In a massively distributed environment and with very large volumes of data, the discovery of the *miki* is very challenging. The conventional and sequential suggested approaches should be carefully designed to be parallelized. Unfortunately, in the literature, there have been no solutions for the problem of parallel discovery of *miki* in massively distributed environments. In the following, we limit our discussion to the popular *ForwardSelection* algorithm [23]. We depict a straightforward parallel solution for it under the spark framework.

Parallel *ForwardSelection* Algorithm: In massively distributed environments, with large amounts of data, extracting *miki* of different sizes is not trivial. Since the *ForwardSelection* algorithm uses a level-wise approach to determine *miki* of size k, its parallel version would perform several k jobs. As a result, with very large volumes of data and a very high size of *miki* to be extracted, the *ForwardSelection* algorithm will give a poor performance. This is due to the high disc I/O access, the candidate approach principle and the comparison step at the reduce phase to emit the *miki* having higher joint entropy. In fact, this does not only lead to a poor performance in terms of execution time but also in terms of data communication cost; *i.e.* with a high *miki* size, the quantity of data being transferred between workers will be very high.

Example 26 *Consider dataset $\mathcal{D}'$ as shown in Figure 3.4. We are looking for determining*

miki of size k which is equal to 2 *in parallel, by using a parallel version of the ForwardSelection algorithm. Suppose that dataset* $\mathcal{D}'$ *is divided into two data splits as presented in Figure 3.8. Each data split (respectively* S_1 *and* S_2*) is processed by a dedicated worker (respectively* w_1 *and* w_2*). At a first Spark job, each worker proceeds by emitting each itemset of size one as a key and its corresponding projection (i.e., combination of the '0s' and '1s') as a value. For instance, the worker* w_1 *would emit (A, 1)* 5 *times.* w_2 *would emit (A, 0)* 3 *times (here a simple optimization can be used consisting of emitting only the itemsets that appear in the transactions i.e. having '1s' projections). Then the reduceByKey phase is in charge of computing the joint entropy of each itemset and emitting the itemset with the highest value of joint entropy. At a second Spark job, the itemset with highest joint entropy is combined to each item in each split to generate the candidate miki list of size 2. After the candidate generation step, the joint entropy of each miki candidate of size two is computed similarly as in the first Spark job. For instance, in our example the first Spark job will make item C as miki of size one (*$H(C) = 1$*). The second Spark job will make the itemset CA as miki of size two (*$H(CA) = 1.905$*) as it has the higher value of joint entropy. This should continue until reaching the miki with size k, i.e. using k Spark jobs. Nevertheless, performing k Spark jobs does not lead to good performance results, particularly when k is not small and when dataset is very large.*

3.5 Conclusion

In this chapter, we have discussed the state of the art about parallel solutions for itemset mining. The main limitations of the existing parallel solutions are the multiple scan of dataset and the memory related issues. Basically, these different limitations become more challenging when the dataset is very large and / or the minimum support is very small or size k of *miki* is very high.

In this first part of the thesis, we address the problem of mining itemsets in parallel. In particular, we handle the problem of parallel mining of $\mathcal{CFI}$s and *miki*. We carry out extensive theoretical and practical studies and propose various solutions validated with very large real-world datasets.

Chapter 4

Fast Parallel Mining of Closed Frequent Itemsets

4.1 Introduction

Data analytics in general, and data mining primitives in particular, are a major source of bottlenecks in the operation of information systems. This is mainly due to their high complexity and intensive call to IO operations, particularly in massively distributed environments. Moreover, an important application of data analytics is to discover key insights from the running traces of an information system in order to improve their engineering. Mining Closed Frequent Itemsets $\mathcal{CFI}$ is one of these data mining techniques, associated with thriving challenges. It allows discovering itemsets with better efficiency and result compactness. However, discovering such itemsets in massively distributed data poses a number of issues that have not been addressed yet by traditional methods. One solution for dealing with such issues is to take advantage of parallel frameworks like MapReduce. In this chapter, we address the problem of distributed $\mathcal{CFI}$ mining by introducing a new parallel algorithm, called Distributed-Closed-Itemset MiningDcim, which uses a new technique of data modeling based on prime numbers. A key feature of Dcim is the deep combination of data mining properties with the principles of massive data distribution.

Our Dcim algorithm will be presented in multiple optimizing stategies in section 4.3. Section 4.4 illustrates our experimental results, in which we carried out exhaustive experiments over real world datasets to demonstrate the efficiency of Dcim for large real world datasets with up to 53 million documents.

4.2 Motivation and Overview

In the past few years, advances in hardware and software technologies have made it possible for the users of information systems to produce large amounts of transactional data.

Although data mining has become a fairly well established field now, its applications in massively distributed environments poses a number of thriving challenges which are a well-known source of bottlenecks for the operation of distributed information systems. This is particularly the case of Frequent Itemset Mining (FIM) [88]. FIM allows discovering important correlations for massive sets of data and reveal key insights for numerous applications, ranging from marketing to scientific data analytics, including the optimization of information systems. Actually, discovering the relationship between features in the running traces of a system, for its optimization, is an active research topic [89, 90].

Unfortunately, mining only frequent itemsets generates an overwhelming number of itemsets. This makes their interpretation almost impossible and badly affects the reliability of the expected results.

Several studies have been conducted to define and generate condensed representations of frequent itemsets in the past few years. In particular, Closed Frequent Itemsets ($\mathcal{CFI}$) [7] have received much attention with very general proposals. Existing algorithms for mining $\mathcal{CFI}$s flag out good performances when the input dataset is small or the support threshold is high. However, when the database increases in size or the support threshold turns to be low, both memory usage and communication costs become hard to bear. Some early efforts tried to speed up the mining algorithms by running them in parallel [91], using frameworks such as MapReduce [9] or Spark [10], which allowed making powerful computing and storage units on top of ordinary machines. In [92], Wang et al. put forward an approach for mining closed itemsets using MapReduce, but it suffered from lack of scalability.

In this chapter, we introduce a new parallel algorithm Dcim for enumerating $\mathcal{CFI}$s using MapReduce. In Dcim, we develop a new approach based on mathematical techniques. The items from the database are transformed into prime numbers, and $\mathcal{CFI}$s are generated by using only division and multiplication operations. When the scale of datasets gets large, such operations could cause an overwhelming computing and memory utilization. To overcome this issue, we propose insightful optimization techniques that enable extracting $\mathcal{CFI}$s from even very large datasets. The main contributions of this chapter are as follows:

1. We propose a numerical representation of transactional datasets using a new transformation technique. This transformation is embedded in the algorithm for a very low additional cost.
2. We design an efficient parallel algorithm for $\mathcal{CFI}$ mining by deeply combining MapReduce functionalities with the properties of $\mathcal{CFI}$s.
3. We exploit the mathematical properties of our numerical representation and provide optimizations both at the architectural level as well as on the computing nodes.
4. We carry out exhaustive experiments on real world databases to evaluate the performance of Dcim. The results suggest that our algorithm significantly outperforms the pioneering algorithms in $\mathcal{CFI}$ mining over large real world datasets with up to 53 million articles.

4.3 DCIM Algorithm

Manipulating string operations causes multiple problems when handling large scale datasets. In fact, when the support threshold turns to be low, both memory usage and communication costs become unbearable. We overcome this issue by designing a distributed solution to mine $\mathcal{CFI}$s using the MapReduce framework. In this section, we propose our DCIM algorithm which distributes the mining process of $\mathcal{CFI}$s over a cluster of nodes by using a number of well specified MapReduce jobs adapted to our mining problem.

4.3.1 Algorithm Overview

The DCIM algorithm uses two MapReduce phases to mine $\mathcal{CFI}$s in three steps which are depicted as follows.

- **Step 1:** *Splitting*: Split T into multiple and successive parts and stores the parts on N different computers. Each part is called a *split*.
- **Step 2:** *Frequency counting*: Execute a first MapReduce job. This step is dedicated to count the support of each item in T and prune non-frequent ones. The output of this step will be a list of items sorted in descending order of supports, and each one is linked with a specific prime number.
- **Step 3:** $\mathcal{CFI}$ *Mining*: This is the key step of DCIM which adopts the second MapReduce pass where the Map phase and the Reduce phase perform different methods. Here, load balancing is a crucial concern and will call for particular care and a comprehensive approach of the distribution principle.

In this chapter, we depict the two last steps of our proposed algorithm, namely the two MapReduce jobs.

Frequency counting

Using a simple MapReduce count process, in this step, DCIM scans the database and computes the frequency of each item. Indeed, the input key-value pair would be like $(key, value = t_i)$, with $t_i \subset T$. For each item, say $i_k \in t_i$, the mapper outputs a key-value pair $(key = i_k, value = 1)$. After all mappers instances are completed, the MapReduce infrastructure feeds the reducers with key-value pairs and the output result is represented as $(key = i_k, value_2 = \Sigma(value))$. Adding the minimum support θ as an input of the job, the set of items is pruned by discarding those which are not frequent and sorted in a descending order of their supports in one list, denoted *Frequency-List*. To proceed with the DCIM algorithm, each item in *Frequency-List* will receive a specified prime number.

$\mathcal{CFI}$ Mining

After generating the *Frequency-List*, sorted in a descending order of supports, DCIM starts the second MapReduce job to extract the complete set of $\mathcal{CFI}$s. We detail the Map and

Reduce phases below. We assume that the mining process of the algorithm is going to be on multiple sub-datasets. At this point, we need to deal with our data and well split the dataset, in order to satisfy the correctness and completeness of our results. To do so, a sub-dataset definition cited in [65] is as follows:

Definition 31 *For a given dataset T, let i be a frequent item in T. i-sub-dataset is the subset of transactions containing i, while all infrequent items, item i and items following i in the Frequency-List are omitted. Therefore, having j as a frequent item in P-sub-dataset, where P is a frequent itemset, jP-sub-dataset is the subset of transactions in P-sub-dataset containing j, while all infrequent items, item j, and items following j in the local Frequency-List are omitted.*

Our splitting process is based on item-based dataset partitioning. In fact, the idea is based on the creation of one split S_i for every θ-frequent item $i \in$ *Frequency-List*. Thus, we extract, for each item, its appropriate sub-dataset. In the Map phase, the algorithm loads the *Frequency-List* of the dataset. In each split from the inputs, the algorithm treats each transaction t_i from split S_i. The input pair is like $(key, value = t_i)$. For each t_i, item i is omitted from the transaction t_i and the remaining items are sorted in descending order of supports by checking the *Frequency-List*. After that, DCIM generates a big integer V_{t_i} representing the transaction by multiplying all the primes representing the items of the transaction. At the end, the Map phase emits item i and the appropriates V_{t_i} as follows $(key = i, value = t_i[1], t_i[2], ..., t_i[n])$ where $n \leq ||S_i||$. Figure 4.2 illustrates the transformation process of our algorithm. Each item is mapped to a prime number (left part of Figure 4.2), while the dataset (on the right) is transformed by a prime number multiplications.

Item	Item's support	Prime number
B	4	2
C	4	3
E	4	5
A	3	7
D	1	11

Figure 4.1: Mapping between items and prime numbers sorted in descendent order of support.

When all mapper instances are completed, reducers read collections corresponding to a group of transactions in the form of big integers representing the sub-dataset linked to the item or itemset in question. Then the mining process begins literally. Before describing the Reduce phase, some properties and definitions are of use in the remainder. Indeed, for every pair of itemsets P and Q represented respectively as two big integers X and Y, the following properties hold.

1. P is a closed itemset extracted from a sub-dataset. P is discovered by concatenating the items having the same support as P (in the sub-dataset)

t_i	Original transactions	Prime numbers.	V_{t_i}
1	A, C	7, 3	21
2	B, C, E	2, 3, 5	30
3	A, B, C, E	7, 2, 3, 5	210
4	B, E	2, 5	10
5	A, B, C, E	7, 2, 3, 5	210

Figure 4.2: Dataset D and its prime number transformation.

2. It is not necessary to develop a sub-dataset of itemset Q included in a $\mathcal{CFI}$ already discovered P, such that the supports of P and Q are equal.

3. $P \subseteq Q$ if the rest of division of Y by X is 0.

In a previous work [93], to facilitate the exploration of sub-datasets and mine $\mathcal{CFI}$s, the authors proposed a new technique that defined a header table which was associated to each context. This table listed the items contained in the corresponding sub-dataset, sorted in a descending order of their supports. However, in our approach, extracting $\mathcal{CFI}$s in the Reduce phase of DCIM does not need the use of this header table, thus avoiding additional process. To do so, we adopt the notion of the Greatest Common Divisor (GCD). Knowing that the GCD of two or more integers, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder, we deduce our closure operator utilizing the following lemma.

Lemma 2 *: Let P-sub-dataset be the subset of transactions containing P. The greatest common divisor in P-sub-dataset represents the closure between all transactions.*

Proof 1 *The closure of an itemset P is produced from the intersection between all transactions containing P. Manipulating prime numbers, the* GCD *between primes is unique. Consequently, having all $V_t i$ from P-sub-dataset, extracting the closure from a set of transactions amounts to calculate the* GCD *between them. Hence, the* GCD *in P-sub-dataset is the closure between transactions composing P-sub-dataset.*

Having the prime number representing the item and its transactions as a set of V_{t_i} as an input for reducers, computing the closure from the sub-dataset is straightforward by computing the GCD of all transactions of the sub-dataset. Doing so, there is no further need to store supports of items contained in the sub-dataset. Indeed, if the closure exists, then it will undoubtedly have the same support as that of the item. By concatenating the closure to the candidate item multiplying the prime number of the item and the number representing the closure, the result of our reduce phase will be a $\mathcal{CFI}$ that is represented as a number added to the set of final results.

Load Balancing

The principles explained above are a strong basis for high performances when mining $\mathcal{CFI}$s. However, fully parallel data mining algorithms has to be deeply combined with the intrinsic characteristics of the distributed framework. We know that the reducers, in MapReduce, cannot start applying the Reduce function before all mappers finish their work. Therefore, when approaching the end of the Map phase, there are usually nodes that are idle waiting for the other ones to finish. It is worth using these nodes for reducing the amount of data that should be transferred from mappers to reducers. The main issue is to find the adequate decomposition of the problem, such that one part of the load may be given to a node that may do some pre-processing and save time to the reducers. This can be done thanks to the nice properties of Gcd, which may be divided into parts of any size. In fact, having a unique Gcd for multiple integers, its computation can be done successively, while maintaining the correctness of the final results. Let us consider that we have n mappers $\{M_1, ..., M_n\}$, and on each mapper i we have $M_{i,k}$ numbers (V_{t_k}) associated to key k. Then we can compute $\text{Gcd}_{i,k}(M_i)$ the local Gcd of mapper i for k on the $M_{i,k}$ V_{t_k} it contains. Later, instead of receiving $\sum_{i=1}^{n} M_{i,k}V_{t_k}$ for key k, a reducer will receive a much lower amount of numbers, corresponding to the results of this pre-computing (n, in the ideal case).

Thus, in Dcim, we anticipate the next step of calculating Gcds, avoiding heavy synchronization, and significantly reducing the computing time by performing a reduce-type function, called *combiner*, before starting the Reduce phase of the proposed algorithm. Doing so, we limit the volume of data transfer between the Map and Reduce tasks. This function runs on the output key-value pairs of the Map phase which are not immediately written to the output and already available in memory. Instead, they will be collected in lists, one list per each key value. Also, in our new algorithm, we set the combiner class as a shuffling class where all instances of Map's output are handled as a set of transactions, represented as a set of V_{t_i}. Actually, for each map output key, the combiner function is called and tries to compute the global Gcd taking V_{t_i}s one by one and applying a series of Gcd calculations between them. It is obvious that, besides the technical tricks, passing summarized Gcds to the Reduce phase of the algorithm enhances the computation and calculation time. The pseudo-code of Map, Combiner and Reduce phases to enumerate $\mathcal{CFI}$s is sketched in Algorithm 1. An example of Dcim running is presented in Figure 4.3.

Algorithm 1: DCIM Algorithm

Input: $\mathcal{D}$ a transactionnal dataset
Output: A complete list of $\mathcal{CFI}$s

Map(i, S_i **)**
- $\mathcal{F} \leftarrow$ the frequency list of items
- $\mathcal{P} \leftarrow$ the prime list associated to frequent items
- **forall** $T_i \in S_i$ **do**
  - // Treating the transactions one by one
  - $\mathcal{T}_i \leftarrow$ OrderItems(T_i)
  - $PT(i) \leftarrow 1$ // Starting V_{T_i}
  - **if** $\mathcal{T}_i \neq \emptyset$ **then**
    - **forall** $j \in \mathcal{T}_i$ **do**
      - //Transforms item j and generate V_{T_i}
      - $PT(i) \leftarrow PT(i) \times \mathcal{P}(j)$
      - **emit** $(\mathcal{P}(j+1), PT(i))$

Combiner(*key:* i, *list*(*values*)*: List-PT(i)* **)**
- $List - Gcd(i) \leftarrow \emptyset$
- $k \leftarrow 0$
- **forall** $PT(i)_k \in List - PT(i), k \leq\mid List - PT(i)$ **do**
  - $Gcd(i) \leftarrow Gcd(PT(i)_k)$ //Computing Gcds from all the transformed transactions
  - $k \leftarrow k + 1$
- $List - Gcd(i) \leftarrow Gcd(i)$
- **emit** *(i, List-Gcd(i))* // For each item the associated GCD from its conditional context

Reduce(*key:* i, *list*(*values*)*: List-Gcd(i)* **)**
- $\mathcal{CFI} \leftarrow \emptyset$
- $Closure(i) \leftarrow \emptyset$
- **for** $Gcd(i) \in List - Gcd(i)$ **do**
  - $Closure(i) \leftarrow Gcd(List - Gcd(i))$ //Computing the closure with the GCD operation
- $\mathcal{CFI} \leftarrow i \cup Closure(i)$
- **emit** *(Null, the hole set of* $\mathcal{CFI}$*)*

Example 27 *Figure 4.2 illustrates how* DCIM *works on dataset* $\mathcal{D}$*. First, having a minimum support* $\theta = 2$*, the frequency counting pass provides the Frequency-List containing items of* T *with their linked primes sorted in a descending order of frequencies (for the same frequencies we apply the alphabetical order on items). Next, the second MapReduce pass of* DCIM *is sketched in Figure 4.3. Starting by the less frequent items from each*

transaction, Dcim *decomposes the* V_{t_i} *to construct sub-datasets. In the example, the first mapper takes {CA} as a transaction. Having* $frq(A) \leq frq(C)$, Dcim *starts by dividing* V_{t_1} *by "7" the prime associated to {A}. The mapper provides {C} as a transaction for A-sub-dataset as a first result. Reciprocally, {A} is provided as a transaction for C-sub-dataset. The same computations are applied for the rest of mappers. Treating {A} as combine inputs in the second table of the example, A-sub-dataset is delivered as a set of* $V_t s$ *(e.g. {C}=3, {BCE}=30, {BCE}=30). With* Gcd= 3 *which is the prime associated to {C}, {AC} is a* $\mathcal{CFI}$*. The same calculations are applied to itemset {AB}, taking into account its sub-dataset as inheritance from A-sub-dataset and so on. The process is stopped in each reducer in two cases: the first case when there is no further item to treat from mappers outputs and the second phase when there is an inclusion relation between a closed itemset found and those provided before the latter.*

Map inputs (V_{t_i})	**Processing V_{t_i}**	**Map outputs (Sub-DS)**
{CA} = {21}	$21 = 3 \times 7$	{A} = 7 : {C} = 3
{BCE} = {30}	$30 = 2 \times 3 \times 5$	{E} = 5 : {BC} = 6
	$6 = 2 \times 3$	{C} = 3 : {B} = 2
{BCEA} = {210}	$210 = 2 \times 3 \times 5 \times 7$	{A} = 7 : {BCE} = 30
	$30 = 2 \times 3 \times 5$	{E} = 5 : {BC} = 6
	$6 = 2 \times 3$	{C} = 3 : {B} = 2
{BE} = {10}	$10 = 2 \times 5$	{E} = 5 : {B} = 2
{BCEA} = {210}	$210 = 2 \times 3 \times 5 \times 7$	{A} = 7 : {BCE} = 30
	$30 = 2 \times 3 \times 5$	{E} = 5 : {BC} = 6
	$6 = 2 \times 3$	{C} = 3 : {B} = 2
Combine inputs (Sub-DS)	**$\mathcal{CFI}$ mining → Reduce outputs**	
{A} = 7 : {3, 30, 30}	Gcd(3, 30, 30) = 3 ⇒ 3 × 7 = 21	
	21 = {AC} ⇒ {AC} is $\mathcal{CFI}$	
{AB} = 14 : {15, 15}	Gcd(15, 15) = 15 ⇒ 14 × 15 = 210	
	210 = {ABCE} ⇒ {ABCE} is $\mathcal{CFI}$	
{AE} ? → {AE} ⊆ {ABCE}	Stop	
{E} = 5 : {6, 2, 6, 6}	Gcd(6, 2, 6, 6) = 2 ⇒ 2 × 5 = 10	
	10 = {BE} ⇒ {BE} is $\mathcal{CFI}$	
{EC} = 15 : {2, 2, 2}	Gcd(2, 2, 2) = 2 ⇒ 2 × 15 = 30	
	30 = {BCE} ⇒ {BCE} is $\mathcal{CFI}$	
{C} = 3 : {7, 2, 2, 2}	Gcd(7, 2, 2, 2) = 1 ⇒ 1 × 3 = 3	
	3 = {C} ⇒ {C} is $\mathcal{CFI}$	

Figure 4.3: Illustrative example of $\mathcal{CFI}$s mining: Map, Combiner and Reduce phases of Dcim

4.3.2 Optimizing Strategies

The load-balancing technique presented above is an issue for obtaining high performances. However, massively distributed data mining applied to very large databases calls for thorough optimizations. In the following, we provide insightful optimizing strategies for improving the performance of DCIM in practice.

Document splitting

Collection frequencies of items can be exploited to Reduce required work by splitting up every document adopting the item-based partitioning approach. The main idea is to observe the transactional dataset and fit each mapper with a group of dependent transactions. Thus, assuming $i \in$*Frequency-List* a frequent item, we can split the document by searching transactions containing i concatenated to other items having the same supports as i and so on. This allows not only having fair splits between mappers, but also decreasing the time complexity of each mapper by pruning transactions not needed to extract the sub-dataset of the item in question.

Multiplying Big Integers

In large datasets, transforming data into numerical forms may generate big integers for which we developed special Multiply operator. Before describing this operator, let us recall some definitions about big integers. A big integer X is handled thanks to its polynomial representation in a given base B as:

$$X = x_0 \times B^0 + x_1 \times B^1 + x_2 \times B^2 + ... + x_n \times B^n$$

where B usually depends on the maximal size of the basic data types, and the coefficients x_i (also called limbs) are basic number data types (such as long or double in Java) and fulfill $0 < x_i < B$.

Due to the format of our final output, we treat base B as a power of 10. It significantly reduces the memory usage of the DCIM algorithm. Given two big integers X and Y in their respective canonical forms as follows:

$$X = \sum_{i=0}^{m}(x_i \times B^i) \quad and \quad Y = \sum_{i=0}^{n}(y_i \times B^i)$$

the big integer $Z = X \times Y$ can be obtained thanks to

$$Z_i = \sum_{k+l=i} (x_k \times y_l)$$

.

Using these basic definitions, for large integers of size n, all the addition, substraction, product and division operations have a complexity of $O(n)$. This means that the number

of basic operations on the basic data storage type is proportional to n. Interestingly enough, for the classical product and division operations, the complexity is $O(n^2)$ for multiplying and dividing two integers of size n. When n becomes big, this cost becomes very handicapping. When handling huge integers, it is then of interest to try to obtain a faster algorithm for multiplication and division operations.There are some solutions proposed to overcome the above-mentioned problem, and we have tried most of them. One of them is the Karatsuba algorithm [94] proposed for an efficient multiplication of big integers. Karatsuba was the first to observe that multiplication of large integer could be made faster than $O(n^2)$. However, its method is a recursive one. It reduces the number of multiplications from the four products $x_0 \times y_0, x_0 \times y_1, x_1 \times y_0$ and $x_1 \times y_1$ to three by dividing the big integers in two parts. To minimize the complexity caused by Karatsuba, a second algorithm called Toom-Cook algorithm was implemented [95]. In fact, Toom-Cook algorithm takes X and Y as two big integers, and splits them into j lower parts, each of length i, and operates on the parts. As j grows, one may mix much of the multiplication sub-processing, hence reducing the overall complexity of the algorithm. The multiplication sub-operations can then be computed recursively using the Toom–Cook multiplication again, and so on. Nevertheless, the complexity of Toom-Cook can be further reduced. Indeed, the product of two large integers of size n can be done in $O(n\log(n))$ thanks to Fast Fourier Transform ($\mathcal{FFT}$) techniques detailed in the following. In fact, two large integers X and Y of size at most $n-1$ can be written in the form of $X = X(B)$ and $Y = Y(B)$, where B is the base (B a power of 10) and X and Y are two polynomials as:

$$X(z) = \sum_{i=0}^{n-1}(x_i \times z^i) \quad and \quad Y(z) = \sum_{i=0}^{n-1}(y_i \times z^i)$$

Denoting by $R(z)$ the polynomial obtained by the product of $X(z)$ and $Y(z)$, we have $XY = R(B)$, and a final rearrangement on the coefficients of R(z) permits to obtain the product XY. As a result, we are led to the problem of multiplying two polynomials of a degree lower than n. A polynomial of a degree lower than n is uniquely defined from its evaluations at n distinct points. Therefore, to obtain the product $R(z) = X(z)Y(z)$, it is sufficient to compute the values $R(w_k)$ at $2 \times n$ distinct points of w_k, which are computing X(w_k) and Y(w_k).The $\mathcal{FFT}$ idea consists in choosing for w_k the complex roots of unity Ω like:

$$w_k = \exp(\frac{2i\Pi k}{2n}) = \Omega^k \quad where \quad \Omega = \exp(\frac{2i\Pi}{2n})$$

Thus, the $\mathcal{FFT}$ algorithm proceeds with a transformation technique called the Fourier transform. For a given sequence $X = (x_0, x_1, ..., x_{2n-1})$ derived from $X(z) = \sum_{i=0}^{n-1}(x_i \times z^i)$, the algorithm computes its Fourier transform F using Ω as follows:

$$F(X) = (f_0, f_1, ..., f_{2n-1}) \ ; \ f_k = \sum_{j=0}^{2n-1}(x_j\Omega^{jk})$$

where the conjugate Fourier transform is :

$$\overline{F}(X) = (f_0, f_1, ..., f_{2n-1}) \qquad with \qquad f_k = \sum_{j=0}^{2n-1} (x_j \Omega^{-jk})$$

Roughly speaking, to compute the coefficients f_k of $F(X)$, the transformation performs the following steps:

1. Define two sub-sequences of size n:

$$X_0 = (x_0, x_2, ..., x_{2n-2}) \ ; \ X_1 = (x_1, x_3, ..., x_{2n-1})$$

2. Compute the Fourrier transform:

$$F(X_0) = (a_0, a_1, ..., a_{n-1}) \ ; \ F(X_1) = (b_0, b_1, ..., b_{n-1})$$

3. Deduce the Fourier Transform $F(X)$ with the formulas:

$$f_k = a_k + \Omega^k b_k \ ; \ f_{n+k} = a_k - \Omega^k b_k \ ; \ 0 \leq k \leq n$$

We now present formally the algorithm to multiply big numbers with $\mathcal{FFT}$ algorithm. Let X and Y be two big integers with less than n coefficients. To compute $Z = X \times Y$ in time $O(n \log(n))$, $\mathcal{FFT}$ performs the following steps:

1. Compute the Fourier transform X' and Y', of size $2n$ each, of the sequences x_j and y_j:

$$X' = (x'_0, x'_1, ..., x'_{2n-1}) \ ; \ Y' = (y'_0, y'_1, ..., y'_{2n-1})$$

2. Compute the product term by term in Z':

$$Z' = (z'_0, z'_1, ..., z'_{2n-1}) \ ; \ z'_i = x'_i \times y'_i$$

3. Compute the inverse Fourier transform Z of Z' with the conjugate $\mathcal{FFT}$ process:

$$Z = (z_0, z_1, ..., z_{2n-1}) \equiv \frac{1}{2n} \overline{F}(Z)$$

Finally, after rearrangement of coefficients z_i, the number

$$Z_i = \sum_{0}^{2n-1} (z_i B^i)$$

is equal to the product of X by Y.

The algorithm consists in computing two $\mathcal{FFT}$s of size $2n$ and one reverse $\mathcal{FFT}$ of size $2n$. As a consequence, the product of two large integers with n digits has a complexity

asymptotically equal to 3 $\mathcal{FFT}$s, say $O(n\log(n)^3)$.

Reducing the size of prime numbers

Dealing with large datasets leads us to efficiently manipulate large numbers. Therefore, in addition to the efficient multiplication operator, we have also tried to reduce the size of generated numbers as much as possible. In fact, when analyzing our execution logs, we have observed that items with a low frequency are much more numerous than those having high support values. Thus, for performance enhancements, we have tried to attribute the lower primes to items that have higher frequencies. This idea has remarkably reduced the running time of our algorithm.

4.4 Experiments

4.4.1 Experimental Setup

To perform our experiments, we use one of the clusters of Grid5000[1] which is a large-scale and versatile test-bed for experiment-driven research on parallel and distributed computing. Our experiments are performed on a cluster with 32 nodes (384 cores in total), equipped with Hadoop 2.6.0 version. Each machine is equipped with a Linux operating system, 96 Gigabytes of main memory, dual-Xeon X5670 with 2.93GHz 12 core CPUs and a 320-Gigabytes SATA hard disk.

Due to lack of parallel $\mathcal{CFI}$ mining approaches in the literature, we compare our algorithm to our own parallel implementation of Closet in MapReduce, the P-Closet. We used three Map Reduce jobs. The first job is dedicated to generate the frequency list containing all items in the dataset and for each one we associate its number of occurrences (support) and the final list is sorted in the descending order of supports. The second job in P-Closet takes the entire dataset and removes all the infrequent items. Eventually, the third job achieves the $\mathcal{CFI}$ mining process. The latter divides the dataset in the Map phase into multiple splits using the item-based partitioning approach mentioned earlier in subsection 4.3.2. The Map phase finds for each frequent item its sub-dataset and the associated header table. The Reduce phase starts by comparing the supports of the items with the supports of the itemsets in the header table of the corresponding sub-dataset. Those having the same supports, their string concatenation produces a $\mathcal{CFI}$ which is stored in a hash-table with its corresponding supports.

Finally, we also compare Dcim to the parallel PFP-Growth [12] implementation of the FP-Growth algorithm (Pfp in short) for MapReduce. Pfp is dedicated to the extraction of frequent itemsets only (and the generation of frequent itemsets from closed frequents ones can be done in a significant amount of time). However, this is an interesting comparison to a well-known approach of the literature. The default values for Pfp in our experiments are:

[1] `https://wiki.inria.fr/ClustersSophia/Clusters_Home`

$Q = 30,000$ (the number of groups containing dependent transactions, for the construction of the corresponding FP-Trees from sub-datasets to each itemset candidates) and $K = 90$ (the number of top frequent itemsets). For more details see [12].

4.4.2 Datasets

We carry out our tests on two real-life datasets. The first one, called "English Wikipedia", represents a transformed set of Wikipedia articles into a transactional dataset, each line mimics an article. It contains around 8,000,000 transactions with almost 7,000,000 distinct items, in which the maximal length of a transaction is 150,000 and the size of the whole database is 4.7 gigabytes. The second dataset, called "ClueWeb", consists of Web pages that were collected in January and February 2009 and has been used by several tracks of the TREC conference. During our experiments, we utilize a part of this dataset with 53 million transactions including 11 million items with a maximal length of a transaction of 700,000. The size of the considered "ClueWeb" dataset is 24.9 gigabytes.

4.4.3 Performence Analysis

Figures 4.4 and 4.6 show the results of our experiments on both English Wikipedia and ClueWeb datasets (respectively). Figure 4.4 reports the comparative performance of Dcim under different values of minimum support (θ) less then 1% of the overall size of the dataset. We see that Dcim sharply outperforms both of its competitors. Actually, the Wikipedia dataset contains a most equally number of items and transactions. Thus, as far as θ value is low, Pfp and P-Closet generate too many candidates, and a lot of long sub-datasets for each one. Indeed, the inclusion tests and evaluations under the pruning methods used in these two algorithms causes lead, as expected, to poor performances. Therefore, the response time of Pfp and P-Closet grows exponentially and goes up quicly. Dcim overcomes these problems by using prime numbers to generate the sub-datasets through division operations. Furthermore, the Gcd in each sub-dataset has eliminated the check of supports between the candidate and its deduced closure, leading to much better performances. For instance, on the Wikipedia dataset, the difference in response time is 5% with a support of $\theta = 60 \times 10^{-3}$, while it grows up to 43% with a support of $\theta = 10 \times 10^{-3}$.

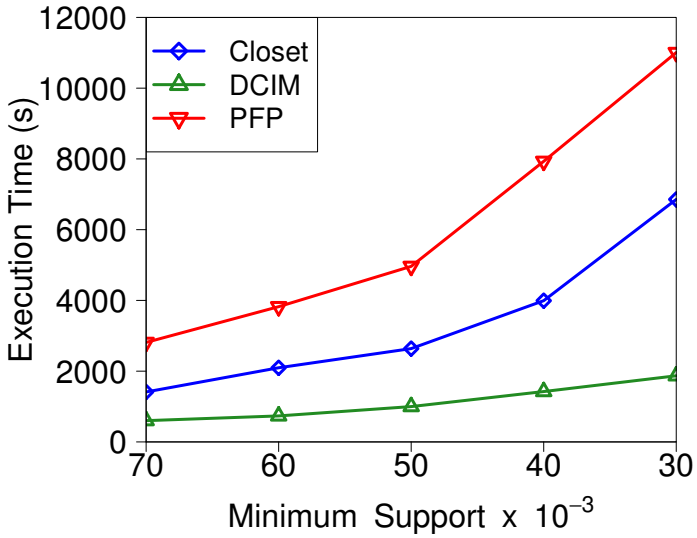


Figure 4.4: Runtime on English Wikipedia dataset with a cluster of 16 nodes: All algorithms

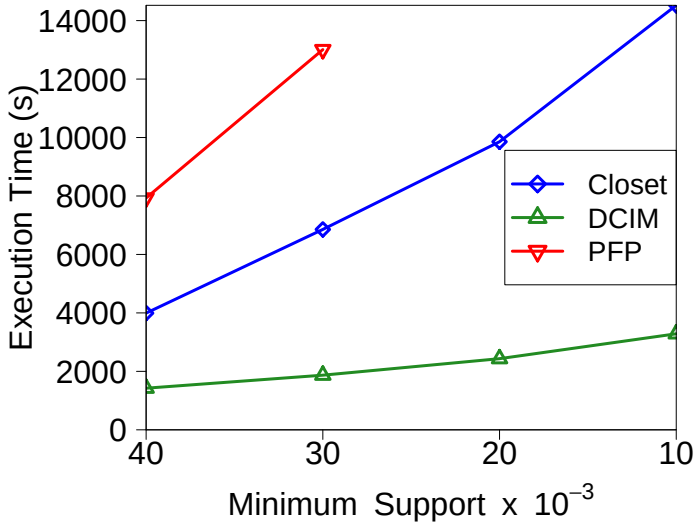


Figure 4.5: Runtime on English Wikipedia dataset with a cluster of 16 nodes: Focus on scalable algorithms

Figure 4.5 highlights the difference between the algorithms of Figure 4.4 that scale. Although Closet continues to scale with $\theta = 40 \times 10^{-3}$, it is outperformed by DCIM, while PFP does not scale for lower threshold values. Furthermore, with $\theta \leq 20 \times 10^{-3}$, we clearly observe a significant difference in the response time between DCIM and all the algorithms

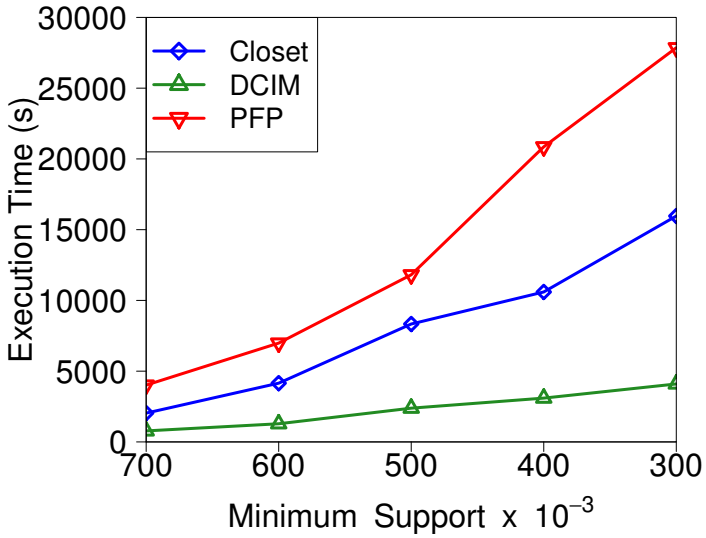


Figure 4.6: Runtime on ClueWeb dataset with a cluster of 16 nodes: All algorithms

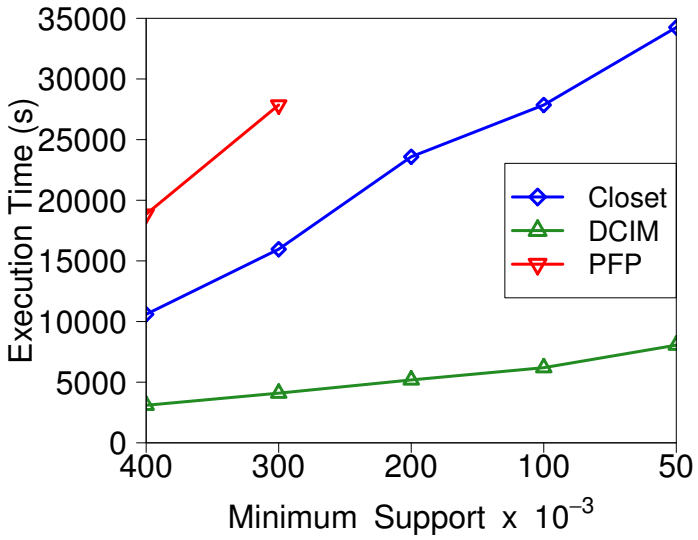


Figure 4.7: Runtime on ClueWeb dataset with a cluster of 16 nodes: Focus on scalable algorithms

from the state of the art, owing to its robust and efficient core mining process.

In Figures 4.6, similar experiments are conducted on the ClueWeb dataset, and we observe very similar behaviors (*i.e.*, DCIM outperforms existing approaches, and the same order between all algorithms is maintained).

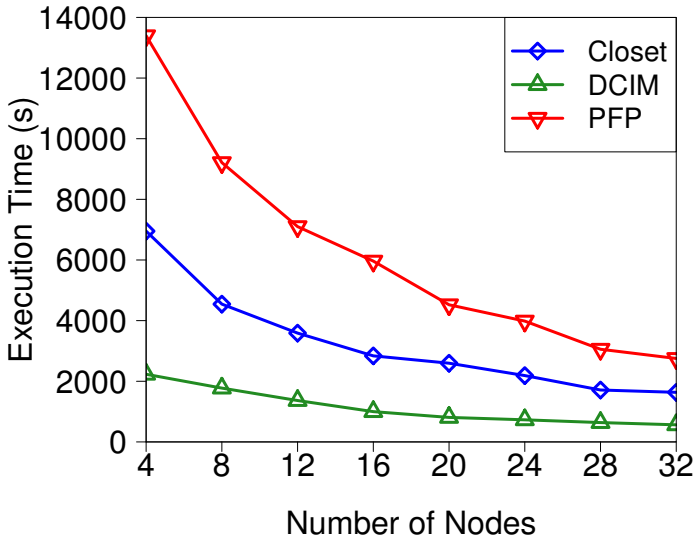


Figure 4.8: Speed-up on English Wikipedia dataset, $\theta = 50 \times 10^{-3}$: All algorithms

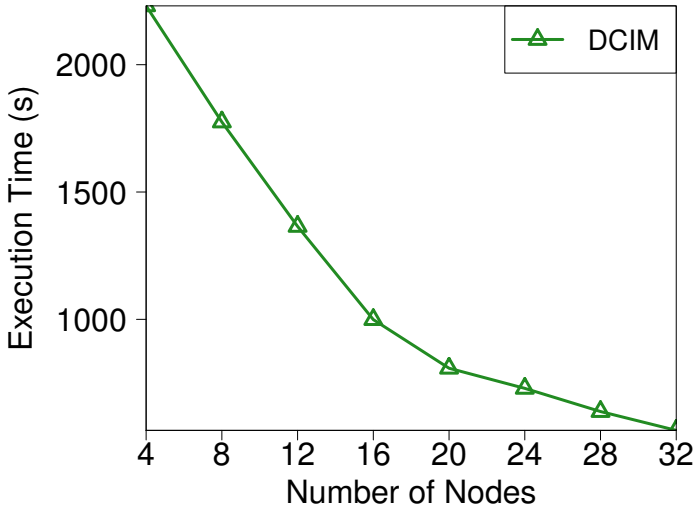


Figure 4.9: Speed-up on English Wikipedia dataset, $\theta = 50 \times 10^{-3}$: Focus on Dcim

Speedup

In order to assess the speedup of our approach, we perform experiments where we measure the response times with a varying number of computing nodes. In the next figures, we perform multiple evaluations over different numbers of nodes, whith $\theta = 50 \times 10^{-3}$, on the

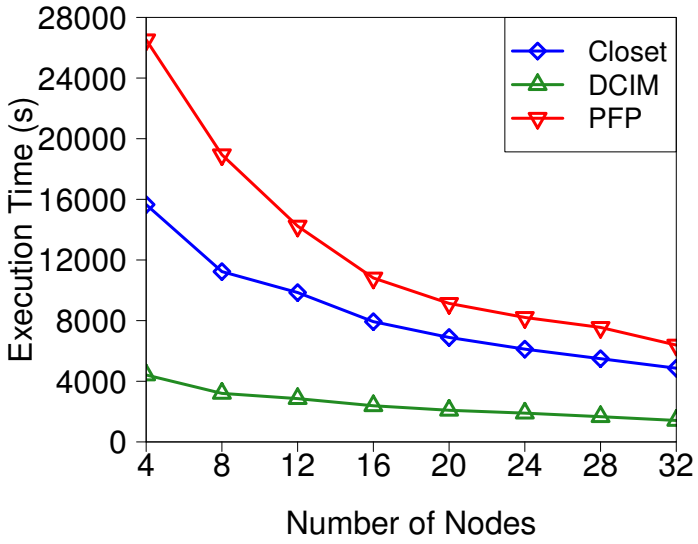


Figure 4.10: Speed-up on ClueWeb dataset, $\theta = 500 \times 10^{-3}$: All algorithms

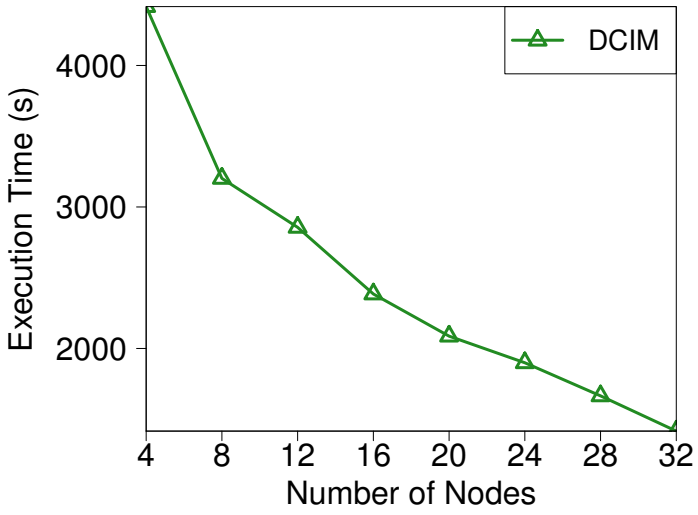


Figure 4.11: Speed-up on ClueWeb dataset, $\theta = 500 \times 10^{-3}$: Focus on DCIM

Wikipedia and ClueWeb datasets. Figures 4.8 and 4.10 show the comparative speed-up results of all algorithms, and confirm the clear advantage of DCIM for all the considered settings in the number of nodes. Figures 4.9 and 4.11 only put the focus on the speed-up of DCIM. This is the same number of nodes and same value of θ (and, of course, the same response times for each number of nodes), with a magnified view on DCIM. We can observe

the very good speed-up of DCIM which, by taking into account parallel optmizations in its core design, benefits from an increase in the number of computing nodes.

4.5 Conclusion

In this chapter, we have proposed a reliable and efficient parallel algorithm for $\mathcal{CFI}$ mining, namely DCIM, which shows significantly better performances than approaches from the state of the art. In addition to using prime numbers and processing big integers, we have provided DCIM with optimizations designed towards massive distribution and the MapReduce framework. The results illustrate that our method outperforms other alternatives, mainly by reducing the overhead of data exchange between nodes.

In the following chapter, we will discuss the problem of mining maximally informative k-itemsets in a distributed environment. More precisely, we will depict the dynamism of data in the context of real-time processing of a database.

Chapter 5

Fast Parallel Mining of Maximally Informative k-Itemsets

5.1 Introduction

In this chapter, we address the problem of mining maximally informative k-itemsets (*miki*) in data streams based on joint entropy. In the last few years, mining *miki*s has been widely studied as a fundamental building block in data analytic and information retrieval. Since [13], only few solutions have scaled. Indeed, when the dataset is massive and/or the k length of the informative itemset to be discovered is high, most of the existing algorithms do not scale. And if they do, the runtime is huge calling for large scale distribution. We propose PentroS, a highly scalable parallel *miki* mining algorithm. PentroS renders the mining process of large volumes of incoming data very efficient. It is designed to take into account the continuous aspect of data streams, particularly by reducing the computations needed for updating the *miki* results after the arrival/departure of transactions to/from a sliding window over simulated streams.

The *miki* problem is formally defined in subsection 3.2.3 of chapter 3. We introduce our PentroS algorithm in detail, in section 5.4. In section 5.5, we evaluate our proposal with massive real-world data streams. Our experimental results confirm the effectiveness of our proposal which allows an excellent throughput with a high itemset length.

5.2 Motivation and Overview

Pattern mining [96] is a core data mining operation and has been extensively studied over the last decade. Recently, mining informative patterns over big data has attracted research interests [97]. Compared with other big data queries, informative pattern mining poses great challenges due to high memory and computational costs, as well as the accuracy requirement of mining results. Such patterns can be itemsets, sequences, sub-trees, or

sub-graphs, depending on the mining tasks and targeting datasets. It has important differences with $\mathcal{FIM}$ [88]. Indeed, the latter puts the focus on discovering frequently occurring patterns from various types of datasets, including unstructured: transaction and text datasets; semi-structured: XML datasets and structured: graph datasets. However, in data analysis [98], frequency of itemsets can not always be an effective measure to give relevant results for a various range of applications, including information retrieval [99]. Indeed, the informativeness of an itemset can result in discovering interesting new patterns that were not previously known.

As stated by information theory [100], the informativeness of one pattern may be computed through its joint entropy. Therefore, the pattern having the highest joint entropy embeds the highest information about the objects in the dataset. Such a pattern is called Maximally Informative k-itemset (*miki*) of length k [13]. *Miki* extraction has been shown to be of interest in many potential applications; for example, it can serve as a basic tool for data mining tasks including classification, clustering, and change detection. Unfortunately, existing *miki* selection algorithms were designed towards static data. Therefore, *miki* mining over data streams becomes an important research topic along with big challenges. Example 28 illustrates a use case of *miki* to retrieve a set of documents over a data stream.

	A	B	C	D	E	
D1	1	0	1	1	1	w1
D2	1	1	0	0	1	
D3	1	0	0	1	1	w2
D4	1	0	1	1	1	
D5	1	1	0	1	1	
D6	0	1	1	1	1	
D7	0	0	1	1	1	
D8	0	1	0	1	1	
D9	0	0	1	1	1	
D10	0	1	0	1	1	
D11	0	1	1	1	1	
D12	1	1	0	1	1	

Figure 5.1: Example of data stream, with records composed of features. Here, the records are documents, and their features are the words they contain.

Example 28 *Consider similarity queries in high dimensional datasets. In this case, we are interested in only using a small subset of the dimensions (or features) for fast record comparisons. Figure 5.1 represents a set of features A, B, C, D, E contained in a stream of documents* $\{D_1, \ldots, D_{12}\}$ *arriving at different time points. In these data, "1" means that the word occurs in the corresponding document, and "0" otherwise. Since a data stream cannot fit into main memory, a usual approach is to consider an observation time*

window that concerns the most up to date data. These are $w1$ and $w2$ in Figure 5.1. At each step, we focus on the data of the latest window only. $w1$ is the first window, at the initialization of the stream. $w2$ is the second one, available afterwards to updates in the stream, and so on. Let us now consider $w1$. DE is a frequent itemset over $w1$, but this information actually provides little help for similarity queries. Given a document q in our data streams, and based on its values of D and E, we will not be able to decide which document is the closest to q. In the meanwhile, itemset ABC is infrequent on $w1$, but much more helpful for this task. With the values {1, 0, 0} (resp. {0, 1, 1}) we can find the corresponding document D_3 (resp. D_6). ABC is a miki of size $k = 3$ over this stream of documents. Our goal is to discover the most up to date miki, continuously and after each update in the stream. For Figure 5.1, this is still ABC in $w2$, but this might not be the case after a few updates.

In the last few years, some new *miki* algorithms have been developed [101, 97]. However, to the best of our knowledge, there is no efficient solution in the literature for parallel miki discovery over data streams. For *miki* mining in data streams, we have to address the following challenges. First, a *miki* mining algorithm needs to explore a search space with an exponential number of candidates. The length of the temporary answer-set itself can be very large. Thus, in a streaming environment, even generating an approximate answer-set can cost much more space than the available one. Therefore, the mining algorithm needs to be very memory-efficient. Second, the computations become more challenging in the presence of high speed data, since we have to quickly extract results and efficiently manage fast sliding window shifts that may affect *miki* candidates.

In this chapter, we propose a parallel solution that deals with the above challenges. We exploit the Spark Streaming framework [10] and propose a clever combination of both information theory and massive distribution principles. We propose a new fast parallel algorithm for computing streaming entropy of *miki*, called PentroS, intended to discover *miki* over data streams, in massively distributed environments. In PentroS, we put forward optimizing strategies to maximize parallelism and to take into account the continuous aspect of data streams. Particularly, we suggest approaches that incrementally update the *miki* results after the arrival and departure of transactions to/from distributed sliding windows.

The remainder of this chapter is organized as follows. In section 5.4, we thoroughly describe our PentroS algorithm. Section 5.5 reports our experimental results over real-world transactional data streams, and section 5.6 concludes the chapter.

5.3 Background

In this section, we formally define the problem, in which we address and sketch the Spark Streaming system as a massively distributed streaming environment.

Data Streams In a data stream, transactions arrive continuously and their volume can be potentially infinite. Formally, a data stream D can be defined as a sequence of transactions, $D = (t_1, t_2, ..., t_i, ...)$, where t_i is the i^{th} arrived transaction. To process and mine data streams, multiple window models are often used. A window is a sub-sequence between i^{th} and j^{th} arrived transactions, denoted as $W[i, j] = (t_i, t_{i+1}, ..., t_j)$ with $i < j$. A user can ask various types of pattern mining questions over various types of window models. The most popular type is the sliding window [102]. Given a sliding window of size w, and a current time point t, we are interested in the continuous pattern discovery in window $W[t - w + 1, t]$. As time changes, the window keeps its size and moves along with the current time point.

5.4 PENTROS Algorithm

5.4.1 Algorithm Overview

Our approach starts with a complete *miki* discovery, from scratch, on the first sliding window, at the initialization of the process. This discovery from scratch might also be done at regular points in the stream whenever incremental computation is not possible. This initial *miki* discovery proceeds in two major rounds. In the first round, it computes local *miki* on each split of the sliding window. Then, it considers the union of all local *miki*s as a set of candidates to be checked over the global distributed sliding window in the second round. To perform the first round, for a given sliding window $\mathcal{SW}$ at a time point t in the stream, presented as a Resilient Distributed Dataset (RDD) containing all the transactions arriving between t_i and t_j with $i < j$, we apply the principles of *Forward-Selection* in parallel on each split of the RDD. The straightforward approach will be to centralize local *miki*s obtained in the first round, and hope to find global *miki* among this set in the second round.

However, this heuristic is optimistic since it considers that global *miki* will appear in at least one split. Actually, it is possible that the global *miki* is never found as a local *miki* in the first round. This is why, in the second round, we need a larger number of candidate itemsets, in order to maximize the chances to obtain the actual *miki*. This can be done by exploiting the set of candidates that are built, locally on each split, in the first round. The last step of *Forward-Selection* aims to compute the projection counting of $|\mathcal{F}| - k$ candidates and then computing their local entropy. Instead of only considering the itemset having the highest entropy, we will emit to an RDD, for each candidate X, its projection counting in the split. The new RDD will include, for each local candidate X_i ($1 \leq i \leq m$, where m is the number of splits), the projection counting of X in a subset of $\mathcal{T}$.

Doing so, the chances to obtain the actual global *miki* in the second round are higher, but it is still possible that a local candidate X has not been proposed in the entire set of splits in the first round. Consider, for example, a biased data distribution, where a

split contains some features with high entropy, and these features have low entropy on the other splits. Then X is proposed in some splits and not in the other ones. Therefore, for each candidate X obtained in the first round, we have two possible cases:

1. X is a candidate itemset on all the splits, so we have its projections in all the splits, and we are able to compute its exact projection counting on $\mathcal{T}$.
2. There is (at least) one split where X has not been generated as a candidate and we are not able to compute its exact projection counting on $\mathcal{T}$.

In the first case, we collect all the necessary information for computing the entropy of X on $\mathcal{T}$ in the second round with no further data scans. The second case is more difficult since X might be the *miki*, but we cannot be sure due to lack of information about its local entropy on (at least) one split. Therefore, in the second round, we need to check the entropy of X on $\mathcal{T}$ by means of a new distributed data scan in order to compute its exact projection counting. The goal of this second round is therefore to check whether no local candidate has been ignored at the global scale. At the end of this round, we have the respective entropy of all the promising candidate itemsets and we are able to pick the one with the highest entropy.

Afterwards, to compute the global projection counting of a candidate in the splits of the sliding window, we proceed as follows. Let W be a sliding window. When W is divided into multiple splits, we have to count for each projection p of an itemset X its corresponding number of occurrences over the entire W. To do so, in a first step, we start by emitting X with its projection p from each split of W (done using the $flatMap$ transformation in Spark Streaming). Then, after the inclusion tests of the projection over transactions, we count the total number of occurrences in all splits. Subsequently, we compute the global counting of the projection (in Spark, this is done by means of the $reduceByKey$ transformation).

The above mentioned approach is the basic version of PENTROS (without optimizations) and is referred to in the remainder as BASIC-PENTROS.

5.4.2 Incremental Entropy Computation

If we locally apply *Forward-Selection* after each update in the data stream, then the algorithm will perform many scans over the sliding windows to compute the entropy of candidates and to find local *mikis*. Actually, let k be the size of the requested itemset and $|F|$ be the number of features in the dataset, the local cost of applying *Forward-Selection* on a split is given by the product of the number of scans and the number of candidates at each scan, *i.e.* $O(k \times |F|)$. This high complexity degrades the performance of local *miki* generation. Therefore, in this section, we propose an efficient approach to significantly reduce the cost of updating candidate entropy. Actually, our approach only needs a unique operation (thus $O(1)$) to update the entropy of an itemset whether a transaction is added to (or removed from) the data stream. It relies on new principles for incremental entropy computation that will allow extremely efficient updates on the entropy of an itemset.

These principles are detailed with Theorems 4 and 5 that facilitate the joint entropy computations. First, we need to reformulate the computation of the entropy of an itemset through Lemma 3.

Lemma 3 *Given itemset X and dataset $\mathcal{T}$ with size n, let $D(\mathcal{T}_X)$ be the set of projections of X in dataset $\mathcal{T}$. For each projection $t \in D(\mathcal{T}_X)$, let f_t be the frequency of t in the dataset. Then the joint entropy of X in $\mathcal{T}$ can be computed as:*

$$H(\mathcal{T}, X) = log(n) - \frac{1}{n} \sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) \tag{5.1}$$

Proof. The total number of projections of X in dataset $\mathcal{T}$ is equal to n. For each projection $t \in D(\mathcal{T}_X)$, let f_t be the frequency of t in the dataset. Thus, the probability of t is $p(t) = \frac{f_t}{n}$. Therefore, we have:

$$H(\mathcal{T}, X) \quad = \quad - \sum_{t \in D(\mathcal{T}_X)} \frac{f_t}{n} \times log(\frac{f_t}{n}) \quad = \quad -\frac{1}{n} \sum_{t \in D(\mathcal{T}_X)} f_t \times log(\frac{f_t}{n})$$

We know that $log\frac{a}{b} = log(a) - log(b)$. Hence, we have:

$$H(\mathcal{T}, X) = -\frac{1}{n}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) - log(n) \times \sum_{t \in D(\mathcal{T}_X)} (f_t))$$

The sum of the frequencies of the projections in $D(\mathcal{T}_X)$ is equal to the total number of transactions in $\mathcal{T}$, i.e. n. In other words, we have $\sum_{t \in D(\mathcal{T}_X)} f_t = n$. Consequently, $H(\mathcal{T}, X)$ can be simplified as:

$$\begin{aligned} H(\mathcal{T}, X) &= -\frac{1}{n}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) - log(n) \times n) \\ &= log(n) - \frac{1}{n}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t)) \square \end{aligned}$$

Entropy Computation After the Arrival of a New Transaction

By introducing Theorem 4, based on our previous Lemma, we propose a very fast computation of the entropy of an intemset after the arrival of a new transaction to the sliding window.

Theorem 4 *Given itemset X and two datasets $\mathcal{T}$ and $\mathcal{T}' = \mathcal{T} \cup \{t'\}$, then, the joint entropy of X in $\mathcal{T}'$ can be computed by using the joint entropy of X in $\mathcal{T}$, and the frequency of $t' \cap X$ in $\mathcal{T}'$, denoted as $f_{t'}$, as follows:*

1. *if $f_{t'} = 1$, then*
   $H(\mathcal{T} \cup \{t'\}, X) = log(n+1) - \frac{n}{n+1}(log(n) - H(\mathcal{T}, X))$

2. *if* $f_{t'} > 1$, *then*

$H(\mathcal{T} \cup \{t'\}, X) = log(n+1) - \frac{1}{n+1}(n(log(n) - H(\mathcal{T}, X)) + f_{t'} \times log(f_{t'}) - (f_{t'} - 1) \times log(f_{t'} - 1)$

Proof. Using Lemma 3, the joint entropy of X in $\mathcal{T}'$ can be written as:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1} \sum_{t \in D(\mathcal{T}'_X)} f_t \times log(f_t) \tag{5.2}$$

Let $t' \cap X$ be the intersection of the new transaction t' and X, and $f_{t'}$ the frequency of $t' \cap X$ in the new dataset $\mathcal{T}'$. Let $\mathcal{T}$ be the old dataset, i.e. the dataset before the arrival of t'. Thus, we have $\mathcal{T}' = \mathcal{T} \cup \{t'\}$. In our proof, we consider two cases : i) $t' \cap X$ exists in $\mathcal{T}$, so $f_{t'} > 1$; ii) $t' \cap X$ does not exist in $\mathcal{T}$, thus $f_{t'} = 1$. In the case where $t' \cap X \in \mathcal{T}$, for updating the entropy for the new dataset $\mathcal{T}'$, then we have to remove the old frequency of $t' \cap X$ (i.e. $f_{t'} - 1$) from the entropy formula, and replace it by the new frequency (i.e. $f_{t'}$). As a result, Equation 5.2 can be rewritten as:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) + f_{t'} \times log(f_{t'}) - (f_{t'} - 1) \times log(f_{t'} - 1))$$

From Lemma 3, we have :

$$\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) = n \times (log(n) - H(\mathcal{T}, X)) \tag{5.3}$$

Hence, in this case, the joint entropy of X in $\mathcal{T}'$ can be rewritten as follows:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1}(n \times (log(n) - H(\mathcal{T}, X)) + f_{t'} \times log(f_{t'}) - (f_{t'} - 1) \times log(f_{t'} - 1))$$

In the second case, where $t' \cap X \notin \mathcal{T}$, we can rewrite Equation 5.2 as follows:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t) + f_{t'} \times log(f_{t'}))$$

Since $f_{t'} = 1$ and $log(1) = 0$, we can simplify the above equation:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1}(\sum_{t \in D(\mathcal{T}_X)} f_t \times log(f_t))$$

Therefore, by using Equation 5.3, the above equation can be rewritten as follows:

$$H(\mathcal{T}', X) = log(n+1) - \frac{1}{n+1}(n \times (log(n) - H(\mathcal{T}, X)))$$

□

By using Theorem 4, after the arrival of a new transaction t' to the sliding window, the entropy of candidate X can be computed simply by using its last entropy (before the arrival of t') and the frequency of $t' \cap X$ (i.e. the intersection of the new transaction and X) in the sliding window. This significantly reduces the cost of updating the candidate entropy in the sliding windows.

Entropy Computation After the Removal of a Transaction

The second challenge in entropy computation appears when removing transactions from the data stream. Indeed, the entropy of an itemset candidate may change when a transaction gets out of the current sliding window. To maintain the correctness of our results with the transactions that leave the sliding window, we propose the following theorem.

Theorem 5 *Given itemset X, dataset $\mathcal{T}$, and a transaction $t' \in \mathcal{T}$, then the joint entropy of X in $\mathcal{T}' = \mathcal{T} - \{t'\}$ can be computed by using the joint entropy of X in $\mathcal{T}$, and the frequency of $t' \cap X$ in $\mathcal{T}'$, denoted as $f_{t'}$, as follows:*

1. *if $f_{t'} = 0$, then*
   $H(\mathcal{T} - \{t'\}, X) = log(n-1) - \frac{n}{n-1}(log(n) - H(\mathcal{T}, X))$
2. *if $f_{t'} > 0$, then*
   $H(\mathcal{T} - \{t'\}, X) = log(n-1) - \frac{1}{n-1}(n(log(n) - H(\mathcal{T}, X)) + f_{t'} \times log(f_{t'}) - (f_{t'}+1) \times log(f_{t'}+1))$

The proof can be done in a similar way as that of Theorem 4.

By using the above theorems, we can update the candidate entropy just by taking into account their intersection with the added/removed transactions.

5.4.3 Reducing the Number of Candidates

The theoretical framework, proposed in the previous subsection, allows us to update itemset entropy very efficiently. However, there are still cases where we need to send candidates to the global entropy counting in the second round of PentroS. Unfortunately, computing the entropy of all *miki* candidates might result in a low response time. This is particularly the case i) for large sliding windows, as it will be illustrated by our experiments in section 5.5; and ii) when the features are not uniformly distributed in the splits of RDDs.

Here, we propose an efficient technique for significantly reducing the number of candidates. The main idea is to compute an upper bound for the entropy of the partially sent

itemsets and discard them if they have no chance to be global *miki*. To do so, we exploit the available information about the *miki* candidates flat-mapped to the second form of the RDD.

Let us describe formally our approach. Let X be a partially sent itemset, and P be a partition that has not sent X and its projection frequencies to the transformed partition $P^{'}$ responsible for computing the entropy of X. In $P^{'}$, the frequency of X projections for a part of the dataset is missing, i.e. in the split of P. We call them *missing* frequencies. We compute an upper bound for the entropy of X by estimating its missing frequencies. This is done in two steps: i) finding the largest subset of X, say Y, for which all frequencies are available; and ii) distributing the frequencies of Y among the projections of X in such a way that the entropy of X is maximized.

To do so, the idea behind the first step is that the frequencies of the projections of an itemset X can be derived from the projections of its subsets. For example, suppose two itemsets $X = \{A, B, C, D\}$ and $Y = \{A, B\}$, then the frequency of the projection $p = (1, 1)$ of Y is equal to the sum of the following projections in X: $p_1 = (1, 1, 0, 0)$, $p_2 = (1, 1, 0, 1)$, $p_3 = (1, 1, 1, 0)$ and $p_4 = (1, 1, 1, 1)$. The reason is that in all these four projections, the features A and B exist, thus the number of times that p occurs in the dataset is equal to the total number of times that the four projections p_1 to p_4 occur. In the second step, let Y be the largest available subset of X in the new partition $P^{'}$. After choosing Y, we distribute the frequency of each projection p of Y among the projections of X that are derived from p. There may be many ways to distribute the frequencies. For instance, in the example of the first step, if the frequency of p is 6, then the number of combinations for distributing 6 among the four projections p_1 to p_4 is equal to the solutions which can be found for the following equation: $x_1 + x_2 + x_3 + x_4 = 6$ when $x_i \geq 0$. In general, the number of solutions for distributing a frequency f among n projections may be huge.

Among all these solutions, we choose a solution that maximizes the entropy of X. The following lemma shows how to choose such a solution.

Lemma 6 *Let $\mathcal{T}$ be a dataset, and X be an itemset. Then the entropy of X in $\mathcal{T}$ is the maximum if the possible projections of X in $\mathcal{T}$ that have the same frequency.*

Proof. The proof is done by implying the fact that in the entropy definition (see Definition 26), the maximum entropy is obtained for the case where all possible combinations have the same probability. Since the probability is proportional to the frequency, then the maximum entropy is obtained in the case where the frequencies are the same. □

The above lemma proposes that for finding an upper bound for the entropy of X (i.e. finding its maximal possible entropy), we should distribute equally (or almost equally) the frequency of each projection in Y among the derived projections in X. Let f be the frequency of a projection in Y and n be the number of its derived projections. If (f modulo n) $= 0$, then we distribute equally the frequency, otherwise we first distribute the quotient among the projections, and then the remainder randomly.

After computing the upper bound for the entropy of X in each sliding window that we handle, we compare it versus the maximum entropy of the itemsets for which we have received all projections (so we know their actual entropy value), and discard X if its upper bound is lower than the current maximum found entropy. This strategy allows PentroS to significantly decrease the number of candidates sent for entropy counting in the second round.

5.4.4 Complete Approach

Algorithms 2 and 3 summarize the main steps of the PentroS algorithm for *miki* discovery over a data stream in Spark Streaming. Algorithm 2 depicts the first job of PentroS over sliding window $\mathcal{SW}$ in data stream $\mathcal{DS}$. The transactions of $\mathcal{SW}$ are partitioned and distributed across multiple nodes (multiple splits S_n). Each node emits its local candidates and their appropriate projections. In case of missing information from at least one node, an upper bound function is executed to estimate the frequency of candidate projections in $\mathcal{SW}$ and a second job is performed to check the accuracy of the obtained results. Algorithm 3 illustrates the steps of our second job in PentroS.

Algorithm 2: PENTROS: First Job

Input: $\mathcal{DS}$ a transaction Data Stream, $\mathcal{SW}$ the length of the sliding windows, k the size of *miki*

Output: *Miki* of size k

flatMap($S_i \in \mathcal{SW}$ **)**
- $\mathcal{F}_i \leftarrow$ the set of features in S_i
- $\forall f \in \mathcal{F}_i$ compute $H(f)$ on S_i //entropy of items
- $TopF \leftarrow max(H(f)), \forall f \in \mathcal{F}_i$
- **while** *not end of the stream and* $i \neq k$ **do**
  - $\mathcal{C}_n \leftarrow$ BuildCandidates($TopF$, $\mathcal{F}_i \setminus TopF$)
  - $\forall c \in \mathcal{C}_p, H(c_i) \leftarrow$ ComputeJointEntropy(c, S_i)
  - $TopF \leftarrow max(H(c)), \forall c \in \mathcal{C}_n$

// $\mathcal{C}_k$ contains all the candidate itemsets of size k
// and $\forall c \in \mathcal{C}_k$, the joint entropy of c is $H(c_i)$
- **for** $c \in \mathcal{C}_k$ **do**
  - $\mathcal{P}_c \leftarrow$ projections(c, S_i)
  - **for** $p \in \mathcal{P}_c$ **do**
    - **emit**($key = c : value = p$)

reduceByKey(*key: itemset c, list(values): projections(c)* **)**
- **if** *c has been emitted by all the workers* **then**

// we have all the projections of c on $\mathcal{SW}$
- $H(c) \leftarrow$ IncrJointEntropy(c,projections(c))
- **emit**(c,$H(c)$) in a file *Complete*
- **else**

// the upper bound of c's joint entropy
  - $Est \leftarrow$ UpperBound(c,projections(c))
  - **emit**(c, Est) in a file "Incomplete"

close(**)**
- $C_{max} \leftarrow$ CandidateWithMaxEntropy("Complete")
- **emit**(C_{max}, $H(C_{max})$) in a file "CompleteMax"
- **for** $c \in$ *"Incomplete"* **do**
  - **if** $Est(c) > H(C_{max})$ **then**
    - // c is potentially *miki*, it has to be checked - **emit**(c,Null) in a file "ToBeTested"

Algorithm 3: PENTROS: Second Job

Input: $\mathcal{SW}$ The sliding windows, *miki* of size k
Output: *Miki* of size k

flatMap(Whole $\mathcal{SW}$ **)**
- Read file 'ToBeTested' from Job1 (once)
- $\mathcal{F} \leftarrow$ set of itemsets in 'ToBeTested'
- **for** $f \in \mathcal{F}$ **do**
  - $p \leftarrow$ projections(f, V_1)
  - **emit** (*key: f, value: p*)

reduceByKey(*key: itemset f,*
list(values): projections(f) **)**
- $H(f) \leftarrow$ IncrJointEntropy(f,projections(f))
- write(f, $H(f)$) to a file "CompleteFromJob2"
- **emit** (*key: f, value: $H(f)$*)

close( )
- // emit miki having highest joint entropy
- Max $\leftarrow$ max("CompleteMaxFromJob1", "CompleteFromJob2")
- **emit**(*miki*,Max)

5.5 Experiments

In this section, we evaluate the performance of PentroS algorithm for *miki* mining through experiments over real-world datasets. In the remainder of the section, we first describe the experimental setup, and then report the obtained results.

5.5.1 Experimental Setup

For comparison, we implement a parallel version of *Forward-Selection* in Spark Streaming. By taking the same default values of experiments, it launches a simple *Forward-Selection* process over each sliding window of streaming. With new incoming and outgoing data, *Forward-Selection* is performed over RDDs with a default distribution, set by Spark Streaming, over multiple splits. In our results, we denote this parallel implementation of *Forward-Selection* as *"ParaForwardSelection"*.

To perform our experiments, we use one of the clusters of NEF[1] which is a test-bed for experiment-driven research on parallel and distributed computing. Our experiments were carried out on a cluster with 32 nodes (384 cores in total), equipped with Spark 1.6.1. Each machine is equipped with a linux operating system, 96 gigabytes of main memory, dual-Xeon X5670 with 2.93GHz 12 core CPUs and 320 gigabytes SATA hard disk.

5.5.2 Datasets

To evaluate the performance of PentroS, we utilize a built-in streaming source of Spark Streaming, fed from two real-life datasets. The first one, called "English Wikipedia", represents a transformed set of Wikipedia articles into a transactional dataset, such that each line mimics an article. It contains 8 millions transactions with around 7 millions distinct items and the size of the whole dataset is 4.7 Gigabytes. The second dataset[2], called "ClueWeb", consists of Web pages that were collected in January and February 2009 and used by several tracks of the TREC conference. During our experiments, we utilize a part of this dataset including 632 million transactions. The size of the considered "ClueWeb" dataset is around one terabyte. For each dataset, we perform a data cleaning task, remove all English stop words from all articles, and obtain datasets where each article represents a transaction and the features are the corresponding words in the article.

In our streaming process, we set our sliding windows parameters to five batches as a window length and the sliding interval at which the window operation is performed to four batches. Each batch is set to 5 seconds of incoming data from "English Wikipedia" and "ClueWeb" datasets.

[1] `https://wiki.inria.fr/ClustersSophia/Clusters_Home`
[2] `http://www.lemurproject.org/clueweb09/`

5.5.3 Performance Analysis

The performances of our algorithms are measured in terms of "throughput". This is the number of transactions that an algorithm is able to process in a batch. Since we have batches of five seconds, the throughput in our case is the number of transactions processed within five seconds.

Both of Figures 5.2 and 5.3 report our experimental results on the whole English Wikipedia dataset. Figure 5.2 reports the performance results for an itemset of size k varying from 1 to 8. We see that the throughput of the *Parallel Forward-Selection* algorithm is very low compared to oits competitors. Above a size of $k = 5$ for *miki*s, the quantity of transactions treated by *Parallel Forward-Selection* converges to 0. This is due to the multiple dataset scans performed in each sliding window to determine an itemset of size k (i.e. *Forward-Selection* needs to perform k rounds for each $\mathcal{SW}$). On the other hand, the performance of the Basic-PentroS algorithm is much better than *Parallel Forward-Selection* and its throughput grows by a factor of 3, and it continues scaling for higher k values. This difference in the performance between the two algorithms illustrates the significant impact of itemset mining in the two round architecture of PentroS. Moreover, by using further optimizing techniques, we clearly see the improvements in the performance. In particular, starting from an itemset having size $k = 8$, we observe a good performance behavior of PentroS compared to Basic-PentroS. By taking advantage of our optimizing techniques, particularly by incremental entropy computations and reducing the number of data split scans, we record an improvement in the throughput of an order of magnitude between PentroS and *Forward-Selection*.

Figure 5.3 highlights the difference between the algorithms that scale in Figure 5.2. Although Basic-PentroS continues to scale with $k = 8$, it is outperformed by PentroS algorithm. With itemsets of size $k = 15$, we clearly observe a significant difference in the response time between Basic-PentroS and PentroS. In Figures 5.4 and 5.5, similar experiments have been conducted on the ClueWeb dataset. We observe that the same order between all algorithms is kept compared to Figures 5.2 and 5.3. In particular, we see that *Parallel Forward-Selection* algorithm suffers from the same limitations as it can be observed on the Wikipedia dataset in Figure 5.2.

Figures 5.6 and 5.7 illustrate the results obtained from running the algorithms using different numbers of nodes (Speed-Up). Figure 5.6 shows the throughput over the "Wikipedia Articles" dataset. The difference in the throughput between all algorithms is maintained while we observe that *Parallel Forward-Selection* does not scale well (it does not benefit from the addition of computing nodes). In Figure 5.7, similar experiments are conducted on the "Clue Web" dataset and the same tendency is maintained for all algorithms, indicating the clear advantage of PentroS.

In Figures 5.8 and 5.9, we show the behavior of the three algorithms over batches of data through streams. We settle the length of a sliding window to 5 batches of 5

seconds, and the sliding interval is four batches. For Basic-PentroS and PentroS, we observe slight decreases in the throughput for some windows. This is due to a scan over the splits in the RDD by launching *Forward-Selection* in the case that an itemset $F - k$ has changed from one window to another. Nevertheless, PentroS keeps the same performance improvements, compared to its competitors.

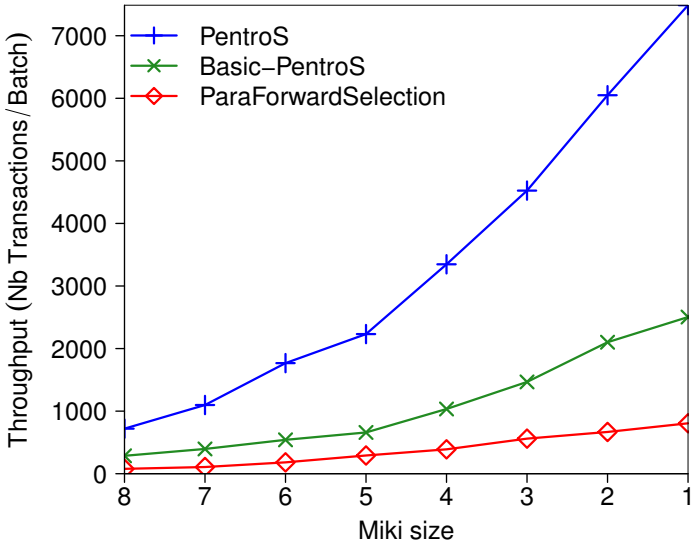


Figure 5.2: Throughput multiplied by 10^{-3} in "Wikipedia Articles" dataset with different values of k (*miki* size): All algorithms

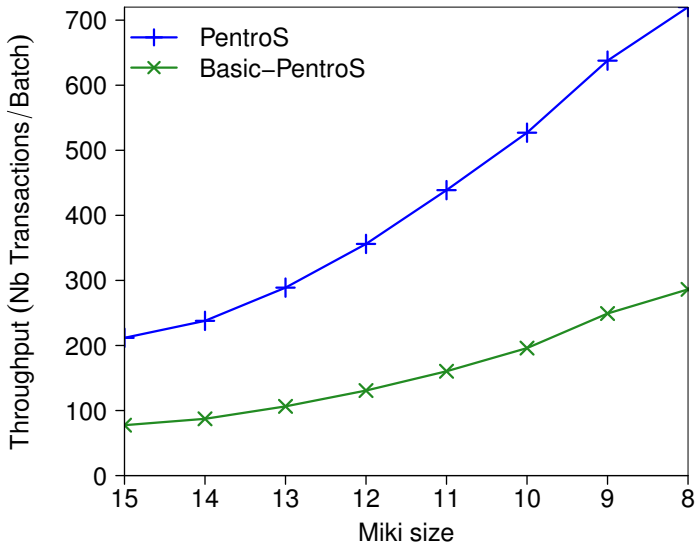


Figure 5.3: Throughput multiplied by 10^{-3} in "Wikipedia Articles" dataset with different values of k (*miki* size): All algorithms: Focus on scalable algorithms

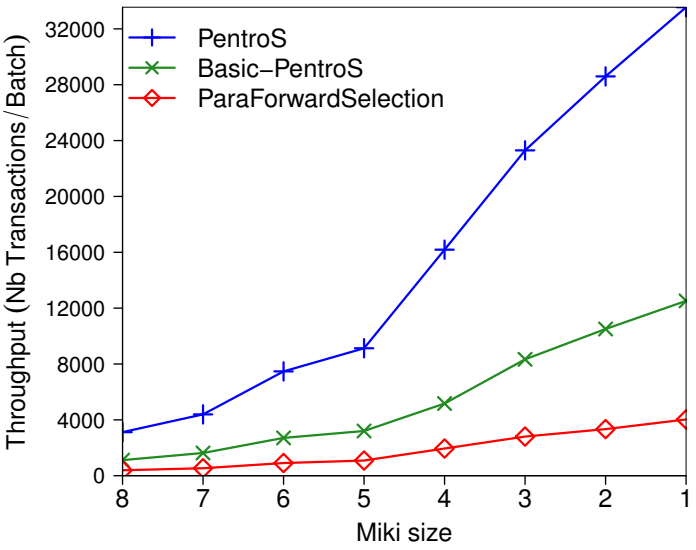


Figure 5.4: Throughput multiplied by 10^{-3} in "Clue Web" dataset with different values of k (*miki* size): All algorithms

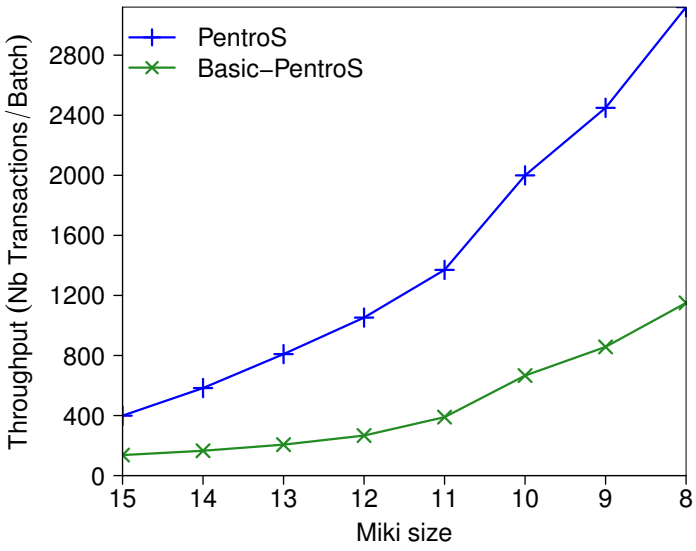


Figure 5.5: Throughput multiplied by 10^{-3} in "Clue Web" dataset with different values of k (*miki* size): Focus on scalable algorithms

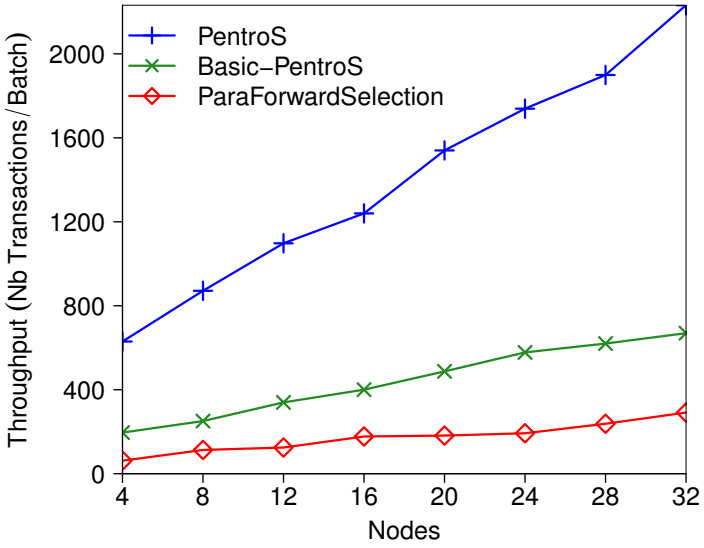


Figure 5.6: Throughput of algorithms by varying the number of nodes and $k = 5$ (size of *miki*): On "Wikipedia Articles" dataset

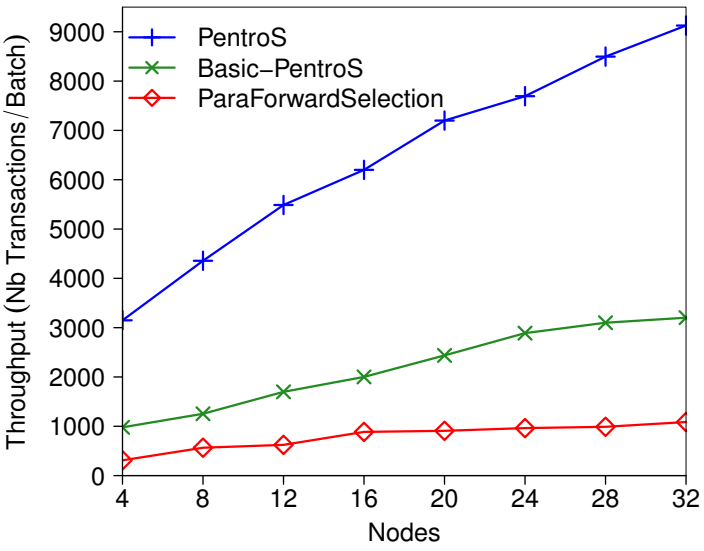


Figure 5.7: Throughput of algorithms by varying the number of nodes and $k = 5$ (size of *miki*): On "Clue Web" dataset

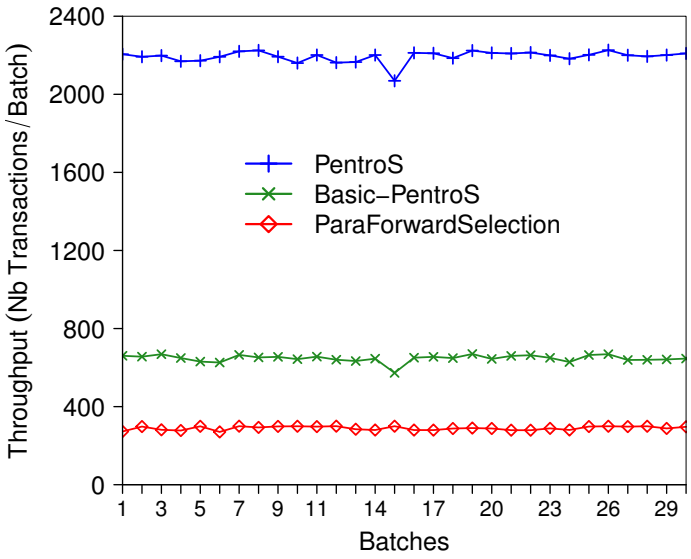


Figure 5.8: Behavior of the algorithms over multiple batches with 32 nodes, and size $k = 5$ (size of *miki*): Over "Wikipedia Articles" dataset.

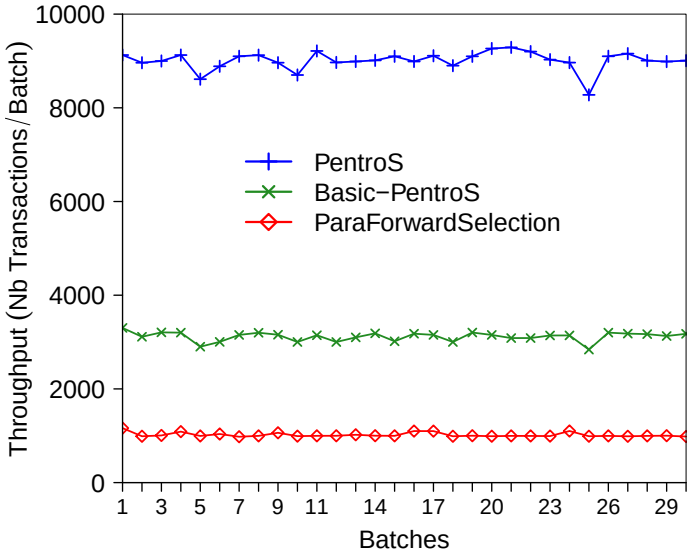


Figure 5.9: Behavior of algorithms over multiple batches with 32 nodes and size $k = 5$ (size of *miki*): Over "Clue Web" dataset.

5.6 Conclusion

In this chapter, we proposed PentroS a reliable parallel streaming algorithm for *miki* mining using the Spark Streaming framework. PentroS has shown a significant efficiency in terms of throughput and scalability. It elegantly determines *miki* over a large number of sliding windows. With PentroS, we propose multiple optimization techniques that render the *miki* mining process very fast. They include the incremental updating of the *miki* results after the arrival and departure of transactions to/from distributed sliding windows. We have extensively evaluated the performance of PentroS using large scale real-world data streams. In short, and in conclusion, the results show the strength and robustness of PentroS in the discovery of *miki*s with a high itemset size over sliding windows with a high rate of incoming and outgoing data.

For the importance of the informations that a *miki* of size k can bring from a dataset, we introduce in the next chapter a use case for *miki*s. Indeed, in the next chapter we prove that the use of *miki* in a classification algorithm can enhance the accuracy rate of new instances classification.

Chapter 6

Fast Parallel Ensemble of Ensembles of Classifiers

6.1 Introduction

Mining maximally informative k-itemsets (*miki*) has shown many challenges. Yet, the utility of informative patterns can be more interesting in classification field. Indeed, classification is one of the building bricks in data mining and information retrieval. The problem has been widely studied in centralized environments ($\mathcal{CE}$). However, in massively distributed environments, parallel classification algorithms have not gained much in terms of accuracy. In this chapter, we address the problem of parallel classification in highly distributed environments. We propose Ensemble of Ensembles of Classifier (EEC), a parallel, scalable and highly accurate classifier algorithm. EEC renders a classification task simple, yet very efficient. Its working process is made up of two simple and compact jobs. Calling to more than one classifier, EEC cleverly exploits the parallelism setting not only to reduce the execution time but also to significantly improve the classification accuracy by performing Two-Level Decision Making (TLDM) steps. We show that the EEC classification accuracy has been improved by using informative patterns and the classification error can be bounded to a small value. EEC has been extensively evaluated using various real-world, large datasets. Our experimental results suggest that EEC is significantly more efficient.

The rest of this chapter is structured as follows. Section 6.3 discusses the related work. Section 6.4 gives an overview of the classification problem, basic used notations, and the necessary background. In section 6.5, we propose our EEC algorithm and we depict its core working process. Section 6.6 reports on our experimental evaluation over various real-world datasets. Section 6.7 concludes the chapter.

6.2 Motivation and Overview

Manually or visually identifying and classifying objects (i.e. instances or examples) is an important requirement of many applications, such as news filtering and organization, document organization, email classification, spam filtering and opinion mining. Classification [14] is a supervised learning process that consists in an automatic assignment of an instance to one (i.e. single-label classification) or more (i.e. multi-label classification) predefined categories, e.g. classes, target attributes or dependent attributes. The classification process turns out to learn a system (i.e. model or classifier) which is capable making good decisions based on its past experience.

Shortly, the classification problem can be defined as follows. Given a training dataset with a fixed number of labeled instances (i.e. each instance has been already assigned to a predefined category), build a model that could classify a new unseen instance to an appropriate category with a small classification error.

Nowadays, we are completely overwhelmed with data coming from different sources such as social networks, sensors, etc. To process these large volumes of data, conventional classifier algorithms have shown their limitations. Typically, data cannot fit into memory, so a classifier cannot learn from large datasets. In addition, classification algorithms are no longer able to efficiently handle large amounts of data in $\mathcal{CE}$.

In this chapter, we propose to distribute the dataset over several machines and to perform different computations in parallel, using parallel framework Spark [103]. For more details about the spark framework please refer to chapter 3, subsection 3.3.2.

So, in a massively distributed environment such as Saprk, except the scalability that can be achieved by a classifier algorithm, there would be no gain in terms of accuracy improvements. For instance, given a training dataset $\mathcal{T}$ that fits into the memory in a $\mathcal{CE}$. Consider a classifier algorithm A in CE trained over $\mathcal{T}$ and a parallel version B of A trained in a massively distributed environment (MDE) using the same dataset $\mathcal{T}$. Logically, regardless to the processing time, the accuracy of A and B would be roughly the same. This means that a naive or a straightforward parallel design of standard (i.e. conventional, sequential or centralized) classifier algorithms would not result in a gain in the classification accuracy, except the improvement in the process run-time. The main reason behind the stationary improvement of the classification algorithms in terms of accuracy between the centralized and the massively distributed environment can be explained by lack of exploiting the full parallelism. In the other hand, variable and feature selection have become the focus of much research in areas of application for which datasets with tens or hundreds of thousands of variables are available. These areas include text processing of internet documents, gene expression array analysis, combinatorial chemistry, etc. The objective of variable selection is three-fold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data.

In the literature, many classifiers perform a data sub-sampling based on a random set of available features. However, it can lead to a lack of accuracy. Therefore, plenty of

feature selection methods are available in literature due to the availability of data with hundreds of variables leading to high dimensionality data. Feature selection methods provides us a way of reducing computation time, improving prediction performance, and a better understanding of the data in machine learning or pattern recognition applications. We propose PAMIKIM, a new algorithm for mining *miki* from big data. As scketched in the experimental section, the use of *miki*s has improved widely the accuracy for the classifier in our learning process. Leading to demonstrate and prove the effectiveness of *miki*s as feature selection techniques.

To summarize, in this chapter, we propose EEC, an efficient, highly parallel classifier algorithm that is able to fully exploit the parallelism settings to improve its classification accuracy. In a massively distributed environment, EEC cleverly increases its classification accuracy, by performing two decision making steps. The first decision step is carried out at a first parallel job taking into account the hole set of *miki*s as attributes, while the second step is done at a second transformation level. The whole working process of EEC is done in two simple, yet efficient jobs. We have evaluated the performance of EEC algorithm through extensive experiments over different real world, large datasets (more than half a billion instances). Our results show that EEC achieves both very good accuracy and execution time compared to other alternatives.

To the best of our knowledge, there has been no prior work on improving the classification accuracy in a massively distributed environments by using *miki*s and the advantages of the full parallelism.

6.3 Related Work

In the literature, there have been several proposed approaches for solving classification problems [24], [25], [26], [27], [28] and [29] to cite a few.

Although their capabilities, each classification technique has its own drawbacks. The performance of the simple instance based K-Nearest Neighbor [104] (KNN) algorithm is linear to its input training dataset. With large number of labeled instances, the time to classify new instances is very expensive (computing the distance of each new instance with all instances in the training dataset).

The Rocchio algorithm [29] has been widely used in text classification [30]. This algorithm accounts for a simple classification technique with a fast learning mechanism, however, it has a low classification accuracy.

Decision Tree (DC for short) [31] has shown better classification performances than KNN and Rocchio algorithms. Indeed, taking the example of text classification with decision tree [105], each term (i.e. word or feature or independent attribute) is assigned to an internal node in the tree. Each branch of the tree is labeled by the weight of the term in the text. The leaf nodes are labeled by the categories (i.e. target variables or classes). DC uses "Divide and Conquer" approach to build a model. Although DC has accounted for good performances, its main limitation is the model size. DC learners can

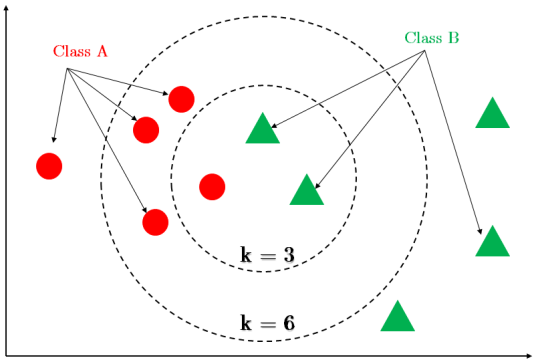


Figure 6.1: KNN classification

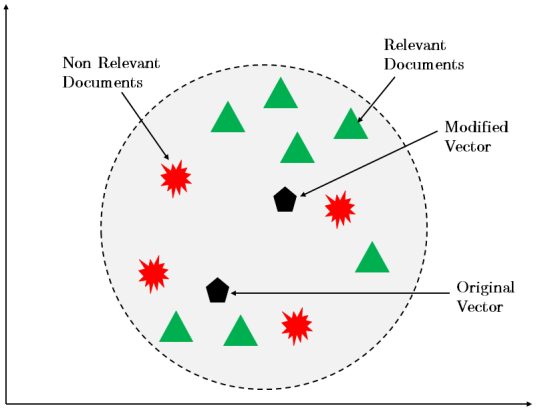


Figure 6.2: ROCCHIO Classification

create over-complex trees that do not generalize well from the training dataset.

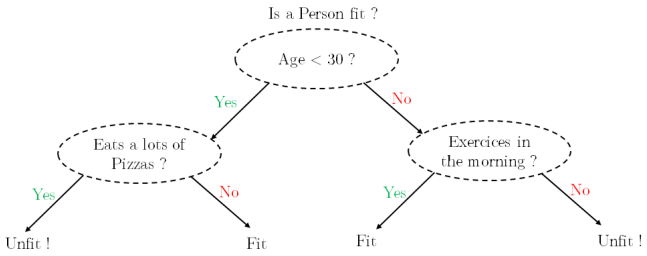


Figure 6.3: Decision Tree Classification

Support Vector Machine (SVM for short) [106], has been successfully applied in classification. It can handle a high dimensional input space. For instance, the task of text classification generally involves a large set of features, SVM can smoothly tackle this problem. The main principle of SVM is to find such linear separators. Although the

SVM approach has demonstrated good performances, its main limitation appears when the number of classes is high. With a large number of classes, the classification task should be divided into several binary classification problems using SVM.

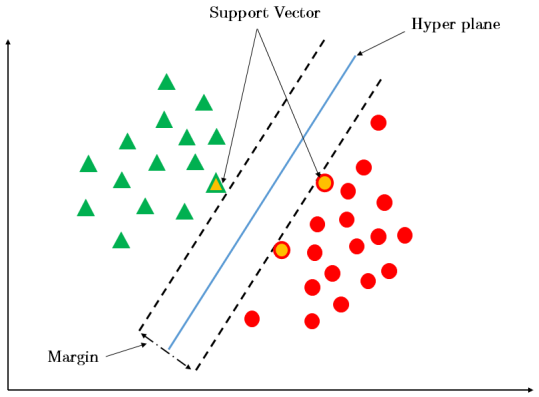


Figure 6.4: SVM Classification

Naive Bayes classifier (NB for short) [34] is the most simple probabilistic classification model. It is based on a strong assumption that all the features in the dataset are independent (i.e. the presence of one feature does not affect other features in the classification). NB computes the posterior probability of each instance with its potential belonging class. NB assigns the instance to the class with the highest posterior probability. In the literature, there have been several endeavors to reduce the strong independence assumption of NB when handling textual data [107]. For instance, the Multinomial Naive Bayes (*MNB* for short), which is an improvement of NB classifier, has shown better performances dealing with text classification problem.

In general, the major challenges in the classification problems have been bounded to the speed up and the accuracy of the models. An interesting technique of Ensemble Classifier (EC for short) [35] has been proposed to improve the classification accuracy. The idea is simple and elegant. Based on a set of base learners (i.e. several classifiers), a decision is made to classify an unseen instance. This technique has been successfully applied with Random Forest (RF for short) [36]. Instead of building a one-tree model, a set of trees is built. To classify a new instance, the decision of all the forest is taken into account. The EC technique has resulted in a significant accuracy improvement of several conventional classification approaches. Moreover, EC has offered the flexibility of various classification techniques to be easily parallelized. However, EC has accounted for some problems, particularly related to the construction of the base learners and their final decision.

With the EC technique, a good classification accuracy can be obtained. The main problem has been the response time of the whole learning and classification process in very large databases. To this end, parallel approaches has been proposed for text classification techniques [37], [36]. For instance, RF has been prallelized in [38]. However, with high

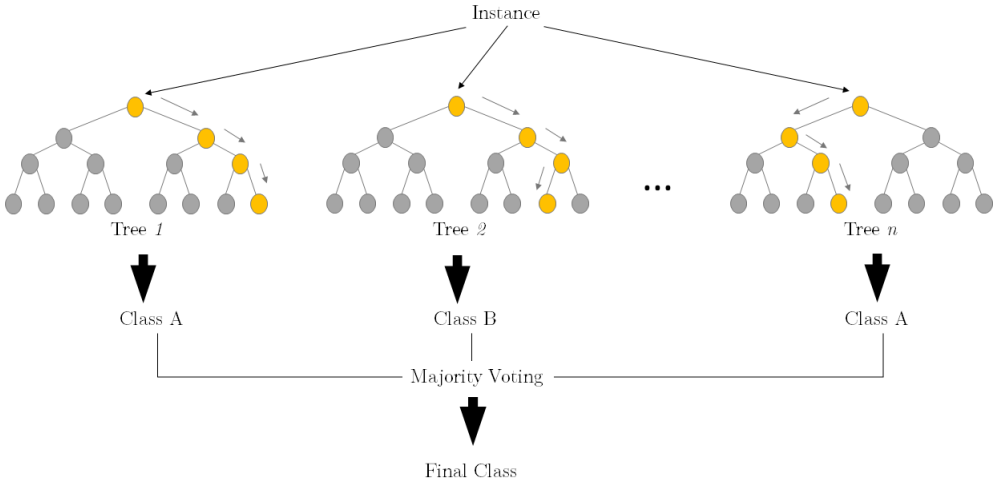


Figure 6.5: RF Classification

number of features, the size of the trees cannot fit into the memory.

In this chapter, we adopt the probabilistic *MNB* classification model, integrate it to a massively distributed and parallel approach for an improved classification accuracy. We introduce our parallel classification technique, namely EEC (Ensemble of Ensembles of Classifiers), based on two decision making steps. EEC is capable to perform the classification task in just two simple, yet efficient Spark jobs, while guaranteeing very high accuracy.

6.4 Background

In this section, first we set up the basic notations and terminology that we are going to adopt in the remainder of this chapter. Second, we introduce the necessary requirements that our work is going to rely on.

6.4.1 Definitions

Some basic definitions are of need for the remainder of this chapter. We start by defining two major terms in our contribution, a training dataset and the classification process over the dataset.

Definition 32 *A Training dataset $\mathcal{T}$ is a set of tuples (i.e. records, rows or instances). Each tuple in $\mathcal{T}$ is described by a set of independent attributes (i.e. features) and a dependent attribute (i.e. class, target attribute or category). Based on the independent attribute values, each tuple is assigned to a target attribute value.*

Definition 33 *Let $\mathcal{T}$ be a training dataset having a set of labeled instances $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ i.e. each instance X_i is assigned to a class Y_j from a set of k fixed number of classes $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_k\}$. The problem of classification is to build a model M, based on the training dataset $\mathcal{T}$, that is capable to return the class Y_j, $j = 1, 2, \ldots, k$ to any new instance previously unseen in $\mathcal{X}$.*

6.4.2 Naive Bayes Classifier

The Naive Bayes (NB for short) classifier is a probabilistic classifier which is based on the Bayes theorem. NB assumes that the features used in the classification are independent from each others. The problem can be formulated as follows. Let $\mathcal{T}$ be a training dataset that consists of n instances $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ where each instance X_i is assigned to a class Y_j from a set $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_k\}$ of k classes. The prior probabilities of each class in $\mathcal{Y}$ are given respectively by $P(Y_1), P(Y_2), \ldots, P(Y_k)$. The posterior probability of a class Y_j occurring for a new instance $X'_{i'}$ previously unseen in $\mathcal{X}$ is proportional to $P(Y_j) \times P(a_1 = v_1 \text{ and } a_2 = v_2 \text{ and } \ldots, a_v = v_v \mid Y_j)$, where $a_1, a_2, \ldots, a_v$ are the features (i.e. independent attributes) having respectively $v_1, v_2, \ldots, v_v$ as values in $X'_{i'}$. By considering the independence assumption between the features, the probability (i.e. posterior probability) that the instance $X'_{i'}$ belongs to the class Y_j can written as: $P(Y_j) \times P(a_1 = v_1 \mid Y_j) \times P(a_2 = v_2 \mid Y_j) \times \ldots, P(a_v = v_v \mid Y_j)$. We compute this product for each value of j from 1 to k and choose the class Y_j with the largest value.

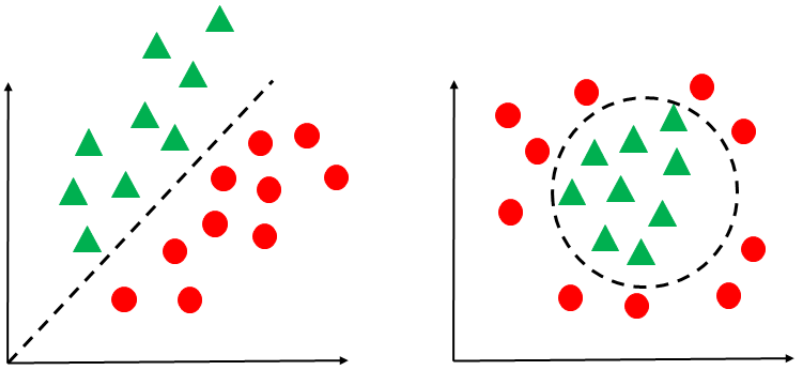


Figure 6.6: NBC Classification

6.4.3 Multinomial Naive Bayes Classifier

Multinomial naive Bayes (*MNB* for short) classifier is a version of Naive Bayes classifier that is widely used in text classification. *MNB* assumes that the features (i.e. words) in a text document follow a multinomial distribution.

Definition 34 *Let $\mathcal{T}$ be a training dataset having a set of labeled documents (i.e. labeled instances) $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ i.e. each document d_i is assigned to a class Y_j from a*

set of k fixed number of classes (i.e. categories) $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_k\}$. *The problem of text classification consists of analyzing the training dataset* $\mathcal{T}$, *to build a model* M *that is capable to give the class* Y_j, $j = 1, 2, \ldots, k$ *to any new instance previously unseen in* $\mathcal{D}$.

MNB classifiers computes the prior probability of all classes Y_j in $\mathcal{Y}$. The prior probability of Y_j is given as follows. $\frac{N_{Y_j}}{n}$ where N_{Y_j} is the number of documents that belongs to Y_j in $\mathcal{T}$ and n is the total number of documents in $\mathcal{T}$. For any new document d', the conditional probability of each feature (i.e. word) W_i in d' is computed as: $CondProb(W_i) = \frac{count(W_i, Y_j)+1}{count(Y_j)+|V|}$

Here, the quantity $count(W_i, Y_j)$ denotes the occurrence number of the feature (i.e. word or term) W_i of d' in the class Y_j. The quantity $count(Y_j)$ denotes the total number of words (with duplication) in the documents that belong to Y_j. $\mid V \mid$ represents the vocabulary in $\mathcal{D}$ (i.e. the number of total distinct words in $\mathcal{D}$).

Eventually, to determine the class of the unseen document d', *MNB* classifier computes the posterior probabilities of each class Y_j in $\mathcal{Y}$ which the product of the prior probability of the class with the conditional probabilities of all words in d'.

The value 1 in the numerator of the conditional probability expression is used to avoid the zero probability when a feature in d' does not exist in the set of features belonging to $\mathcal{D}$.

6.4.4 Features Selection

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for four reasons : simplification of models to make them easier to interpret by users, shorter training times, to avoid the curse of dimensionality and enhanced generalization by reducing overfitting (formally, reduction of variance). The central premise when using a feature selection technique is that the data contains many features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information [108]. Redundant or irrelevant features are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

In what follows, we propose our method for feature selection issued from pattern mining technics [96] which are a core data mining operations and has been extensively studied over the last decade as showed in previous chapters. In this chapter, we propose that the set of *miki*s can be used as a basic tool for data mining tasks including classification, clustering, and change detection.

The *miki* problem is formally defined in subsection 3.2.3 of chapter 3. In the following section, we detail our EEC algorithm.

6.5 EEC Algorithm

In the literature, most of proposed classifier algorithms involve either complex techniques that have ended up in a higher accuracy at the expense of slow runtime, or simple techniques that have resulted in a fair accuracy with a fast learning process. This trade-off is even more important when the training dataset tends to be very large (e.g., billion of data instances). Thus, the major challenge is to build a parallel classification model that is able to achieve both high classification accuracy as well as reasonable processing time.

We take full advantage of the parallelism to improve both the execution time at the learning process and the overall classifier accuracy (which is not the case for other classification algorithms where the same training dataset gives constant accuracy in centralized and distributed environments). We propose EEC which is built based on two Spark jobs. The first job is dedicated to locally training several classifiers at each single partition of the RDD. The second job is dedicated to cleverly classify new instances using a specific majority voting scheme. Interestingly, majority voting in the case of distributed environments calls for a very cautious theoretical analysis. In fact, this mechanism, as we design it for the massively distributed case, allows for very important gains in response time. However, depending on the data distribution, a result may be wrong. We provide the necessary theoretical analysis to show the correctness of EEC.

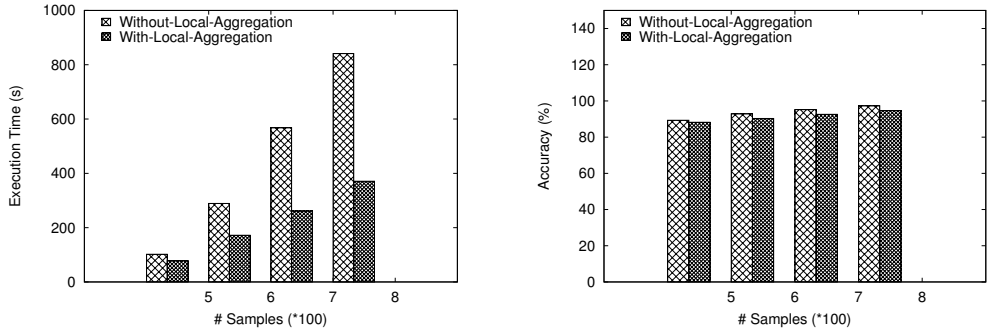
In the following, we thoroughly detail the core working process of EEC algorithm. We first introduce the main features that EEC relies on at the first Spark job (i.e. training step) for improving classification accuracy. Then, we introduce our specific majority voting technique (at the classification step) that allows faster execution of EEC. Eventually, we introduce the complete working process of our EEC algorithm.

6.5.1 Local Base Learners

An ensemble of classifiers can achieve better performance than a single classifier [35]. In a massively distributed environment, creating an ensemble of classifiers can be done efficiently since the data is naturally distributed among computing nodes. Due to the availability of large resources (i.e. memory, CPU etc.) it is possible to increase the number of the ensemble of classifiers in order to improve the overall classification accuracy. To this end, at the learning step (i.e. first Spark job), EEC takes these resources advantages to create k *MNB* local classifiers at each RDD split.

EEC uses features from *miki*s (a bunch of attribute values (all values of each attribute) with their class labels to create k training data sub-samples. These data sub-samples are created by selecting features from the set of maximaly informative k itemsets, mined in a pre-processing step, at each split k times. This process results in a set of k data sub-samples at each split. A *MNB* classifier independently learns from these k data sub-samples. This learning procedure yields in k independent *MNB* classifiers at each split. Depending on the available number of computing nodes, EEC will have an elastic strategy on the number of created training data sub-samples.

6.5.2 One Level Decision Making (OLDM)



(a) Classification Runtime (b) Classification Accuracy

Figure 6.7: 2-Levels Decision Making vs. 1-Level Decision Making

Let $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$ be a set of workers, and $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ be a trained set of *MNB* classifiers generated from EEC's first job. At its second Spark job, EEC divides $\mathcal{B}$ into k disjoint data partitions, $\mathcal{B}' = \{p_1, p_2, \ldots, p_k\}$, where each p_i in $\mathcal{B}'$ is assigned to a split m_i in $\mathcal{M}$. For each new instance X to be classified, X is tested against each single classifier in p_i at each split (i.e. partition of $\mathcal{B}'$). The classification result of each single *MNB* classifier at each split is directly emitted to the worker. Then, the worker aggregates the results and assigns the class having the higher number of votes to X . This voting process of the ensemble classifiers at the REDUCEBYKEY step is a one level decision making (OLDM), where the final assignment decision is carried out at one level (i.e. at the REDUCEBYKEY step). Although this leads to an improvement of the classification accuracy, as illustrated in our experiments, this approach is not optimal in execution time, since the performance of the REDUCEBYKEY phase would impact the overall classification process. To this end, our main challenge is to reduce the overall computing time without degrading the classification accuracy. To that end, we deeply combine the properties of *MNB* to *i)* a theoretical study of the error bounds in the case of local majority voting; and *ii)* the design of the distributed classification algorithm.

6.5.3 Two Level Decision Making (TLDM)

To handle the limitations of the one level decision making step and improve the execution time, EEC relies on a second level of classifier's decisions. At each split of the second job of EEC algorithm, a local decision is made to classify an unseen instance X. To that end, at each worker, a majority voting scheme is carried out to determine the most appropriate class where X should be assigned. Thus, each split would emit one decision (*i.e.*, a class assignment) to the REDUCEBYKEY step. Likewise in OLDM, the REDUCEBYKEY step simply aggregates the majority of votes received from all the workers) and emits the instance with its highest voted class. The two level decision making (TLDM) technique

would ensure both an improvement in the overall classification accuracy and an optimization in the runtime process. In particular, The runtime improvement gain is explained by the local aggregation of the classifier's votes at each worker. We confirm our intuition experimentally by comparing the classification speed up and the classification accuracy of both OLDM and TLDM. To this end, as a simple example we use the 2005 TREC Spam dataset [109]. Figure 6.7 illustrates the difference between performing a classification task based on OLDM and TLDM. Figure 6.7(b) focuses on the accuracy performance of both techniques. By increasing the number of sub-samples from 5 to 8, we observe that the accuracy of TLDM is better than OLDM Figure 6.7(a), clearly illustrated that the execution time of TLMD is better than OLDM. This difference in the runtime performance is simply explained by the reduction in the shuffling time, since a local aggregation has been made in the workers, there are few data transmitted from the workers to the REDUCEBYKEY step. This simple experiment motivates our adoption of TLMD to improve the overall classification process of our EEC algorithm. However, despite the gain in the performance, the two levels decision making technique may lead to some error in the final results, because of performing a majority voting scheme at each split independently. In the following, we show that this error can be bounded to a small value which renders the EEC algorithm feasible.

Lower and Upper Bounds Error

Let us consider a text classification problem with two classes (A and B). Figure 6.9(a), shows the votes of a set of 50 classifiers on one text document. Let us consider a distributed environment with 5 workers where the classifiers are distributed. In the case of a horizontal partitioning of the 50 classifiers of Figure 6.9(a), (*i.e.*, the first two lines of the table are given to split 1, then lines 3 and 4 are given to split 2, etc.), there will be no error in the final vote. Actually, each split will vote for B as a majority class and B will be the final result for this document. However, depending on the partitioning, it is possible that considering only majority votes does not correspond to the actual vote of the whole set of classifiers. Figure 6.9(b) shows a different partitioning of the exact same set of classifiers. In this case, three workers will have A as a majority vote, making it the selected class in the end. We analyze the bounds of the error that can happen leading to a minority class being returned as the majority one at the final classifiers voting.

Let d be a document (i.e. an instance) to be classified. Let $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ be a set of *MNB* classifiers generated from the first job of EEC, and $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ be the set of classes. Let $O(c_i, \mathcal{B})$ be the set of classifiers in $\mathcal{B}$ that assigned d to a class c_i. Let $v_i = |O(c_i, \mathcal{B})|$ be the number of occurrences of c_i in $\mathcal{B}$ (*i.e.*, v_i is the number of classifiers that assigned instance d to class c_i). Let $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$ be a set of workers. Let $\mathcal{B}' = \{p_1, p_2, \ldots, p_k\}$ be a representation of disjoint partitions of $\mathcal{B}$ where each partition p_i is assigned to a split m_i. Let c_x be the class having the lowest number (v_x) of assigned instances (*i.e.*, $\forall i \in \{1..m\}, i \neq x, v_i \leq v_x$).

The following lemma 7 shows a lower bound on the number of c_x in $\mathcal{B}$ for an error to happen in $\mathcal{B}'$.

Lemma 7 *Given $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ a set of n Bayesian classifiers and $\mathcal{B}' = \{p_1, p_2, \ldots, p_k\}$ their representation as k disjoint data partitions. Let v_x be the number of a minority voted class c_x in $\mathcal{B}$. An error in the final voting can happen only when the following condition holds.*

$$v_x \geq \left\lfloor \frac{k}{2} + 1 \right\rfloor \times \left\lfloor \frac{n}{2k} + 1 \right\rfloor \tag{6.1}$$

Proof 2 *An error can happen in the final voting when c_x is a majority class in at least $\lfloor \frac{k}{2} + 1 \rfloor$ partitions of $\mathcal{B}'$. Knowing that an error happens in a single partition p_i in $\mathcal{B}'$ when the number of c_x is greater than 50% in p_i (i.e. the number of c_x is greater than the size z of p_i). The size z of a partition p_i is $z = \lfloor \frac{n}{k} \rfloor$. Therefore, to have an error in a partition, the number of c_x in that partition should be at least $\lfloor \frac{n}{2k} + 1 \rfloor$. Thus an error in the final voting by using $\mathcal{B}'$ can happen when we have:*

$$v_x \geq \left\lfloor \tfrac{k}{2} + 1 \right\rfloor \times \left\lfloor \tfrac{n}{2k} + 1 \right\rfloor$$

□

Lemma 7 determines the lowest number of the minority voted class in $\mathcal{B}$ that may cause an error in $\mathcal{B}'$. We proceed by determining the maximum number of a minority voted class in $\mathcal{B}$ that can result an error in $\mathcal{B}'$. This number is simply computed by considering the total number of the classifier votes (n) in $\mathcal{B}$. This maximum should be less than 50% of classifier votes minus one in $\mathcal{B}$ (otherwise, that class would not be a minority one).

Theorem 8 *Given $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ a set of n Bayesian classifiers and $\mathcal{B}' = \{p_1, p_2, \ldots, p_k\}$ their representation as k disjoint data partitions (i.e. workers). Let v_x be the number of a minority voted class c_x in $\mathcal{B}$. An error may happen in $\mathcal{B}'$, i.e. c_x becomes a majority voted class, when:*

$$v_x \in \left[\left\lfloor \frac{k}{2} + 1 \right\rfloor \times \left\lfloor \frac{n}{2k} + 1 \right\rfloor, \left\lfloor \frac{n}{2} - 1 \right\rfloor \right] \tag{6.2}$$

Proof 3 *Lemma 7 implies that the lowest number of v_x in $\mathcal{B}$ that could result in an error in $\mathcal{B}'$ is $\lfloor \frac{k}{2} + 1 \rfloor \times \lfloor \frac{n}{2k} + 1 \rfloor$. However, the final result is considered as an error only when c_x in $\mathcal{B}$ is a minority class. Hence, v_x must be less than the half of the classifier votes minus one.* □

Based on Theorem 8, in the worst case, the percentage of the cases that can lead to an error in $\mathcal{B}'$ can be computed as follows.

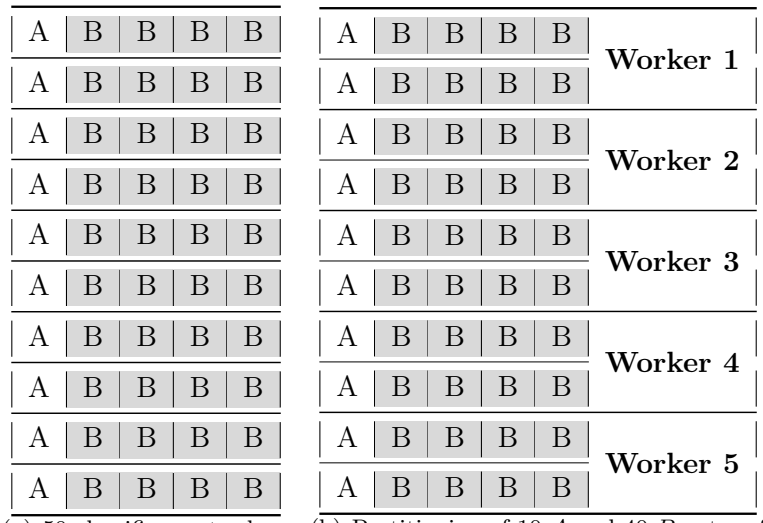
Corollary 1 *Given $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ a set of n Bayesian classifiers and $\mathcal{B}' = \{p_1, p_2, \ldots, p_k\}$ their representation as k disjoint data partitions. Let v_x be the number of a minority voted class c_x in $\mathcal{B}$. The percentage of the cases that may lead to an error in $\mathcal{B}'$ can be computed as follows:*

$$ErrorCases = \frac{\left\lfloor \frac{n}{2} - 1 \right\rfloor - \left\lfloor \frac{k}{2} + 1 \right\rfloor \times \left\lfloor \frac{n}{2k} + 1 \right\rfloor + 1}{n+1} \times 100 \tag{6.3}$$

Proof 4 *Based on Theorem 8, an error can be occurred in $\mathcal{B}'$, when v_x is in the following interval:*

$$\left[\left\lfloor \frac{k}{2}+1 \right\rfloor \times \left\lfloor \frac{n}{2k}+1 \right\rfloor, \left\lfloor \frac{n}{2}-1 \right\rfloor\right]$$

The difference between the upper and lower bound of this interval gives the number of cases that lead to an error in $\mathcal{B}'$. To obtain the percentage of errors, we can simply divide the number of error cases over the total number of values that v_x can take, i.e. $n+1$ cases since the value of v_x should be in the interval $[0 \dots n]$. □



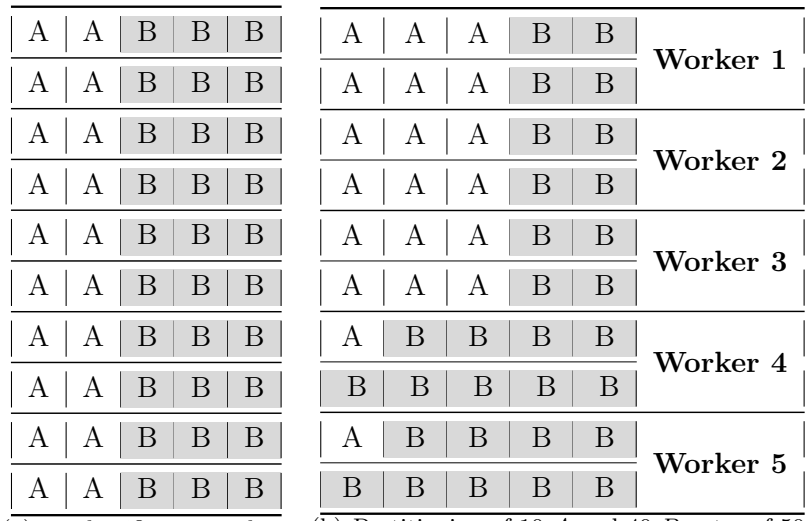
(a) 50 classifiers votes between A and B

(b) Partitioning of 10 A and 40 B votes of 50 classifiers among 5 workers

Figure 6.8: (Left) The distribution of A and B votes of 50 classifiers. (Right) The distribution of A and B votes divided between 5 workers

Example 29 *Figure 6.8 illustrates a case where an error cannot happen with a specific classifier votes distribution. Figure 6.8(a) shows 50 Bayesian classifiers $\mathcal{B} = \{b_1, b_2, \dots, b_{50}\}$, where 10 classifiers voted for a minority class A and 40 classifiers voted for a majority class B. Figure 6.8(b) shows 5 workers $\mathcal{M} = \{worker_1, worker_2, \dots, worker_5\}$ where each split holds a disjoint data partition of $\mathcal{B}$. Clearly the minority class in 6.8 is A while the majority class is B. The number of minimum v_x of A that must be in $\mathcal{B}$ to make a mistake in $\mathcal{M}$ is: $v_x = \left\lfloor \frac{5}{2}+1 \right\rfloor \times \left\lfloor \frac{50}{2*5}+1 \right\rfloor = 18$. Likewise, to make an error in $\mathcal{M}$, the maximum number of v_x in $\mathcal{B}$ is $v_x = \left\lfloor \frac{50}{2}-1 \right\rfloor = 24$. In Figure 6.8, we have $v_x = 10 < 18$, thus an error cannot happen in $\mathcal{M}$.*

Figure 6.9 illustrates the case where an error can happen with a specific classifiers distribution in the workers. Figure 6.9(a) shows 50 Bayesian classifiers $\mathcal{B} = \{b_1, b_2, \dots, b_{50}\}$,



(a) 50 classifiers votes between A and B

(b) Partitioning of 10 A and 40 B votes of 50 classifiers among 5 workers

Figure 6.9: (Left) The distribution of A and B votes of 50 classifiers. (Right) The distribution of A and B votes divided between 5 workers

where 20 *classifiers voted for a minority class* A *and* 30 *classifiers voted for a majority class* B. *Figure 6.9(b) shows* 5 *workers* $\mathcal{M} = \{worker_1, worker_2, \ldots, worker_5\}$ *where each split holds a partition of* $\mathcal{B}$. *The minimum value of* v_x *for* A *to make a mistake in* $\mathcal{M}$ *is then equal to* $v_x = \left\lfloor \frac{5}{2} + 1 \right\rfloor \times \left\lfloor \frac{50}{2*5} + 1 \right\rfloor = 18$. *Likewise, to make an error in* $\mathcal{M}$, *the maximum number of* v_x *in* $\mathcal{B}$ *is* $v_x = \left\lfloor \frac{50}{2} - 1 \right\rfloor = 24$. *The percentage number of cases for* v_x *that may lead to an error in* $\mathcal{M}$ *is*

$$ErrorCases = \frac{\left\lfloor \frac{50}{2} - 1 \right\rfloor - \left\lfloor \frac{5}{2} + 1 \right\rfloor \times \left\lfloor \frac{50}{2*5} + 1 \right\rfloor + 1}{50+1} \times 100 = 13\%$$

Notice that this error percentage is computed for the worst case, and in the average case it is much smaller. Thus, it wont compromise the accuracy of our approach, as shown by our experimental results in Section 6.6.

6.5.4 EEC: Complete Approach

In this section, we introduce our complete approach of EEC algorithm 4. We thoroughly depict its learning and classification processes. In particular, we detail the working principle of each worker of EEC.

Job1: Learning

Given a training dataset $\mathcal{T}$ of labeled documents, at its first Spark job, EEC proceeds by learning a set of $k \times n$ local *MNB* models, where n is the total number of splits and k is

Algorithm 4: EEC

//Job1: Learning
Input: Training dataset $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, k: Number of sub-samples
Output: $n \times k$ *MNB* local classifiers
//FlatMap Task 1
flatmap(*key: Null*: K_1, *value* = Whole input split :t_i V_1 **)**
- *counter* = 1
- **while** *counter* $\leq k$ **do**
  - Sub-sampling V_1 by choosing a set of features from *miki* set
  - Build an *MNB* classifier
  - **emit** (*key: Classifier ID, value: (Feature, Class Label, Prior probability, feature Likelihood*) - *counter* $\leftarrow$ *counter* + 1

//ReduceByKey Task 1
reduceByKey(*key: Classifier ID*, *list*(*values*) **)**
- **while** *values.hasNext*() **do**
- **emit** (*key:(Classifier ID) values.next (value)*)

//Job2: Classification
Input: Set of *MNB* $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$, document d
Output: d with its class label
//FlatMap Task 2
- Read d from a hadoop distributed cache

flatmap(*key: Null*: K_1, *value* = Whole set of *MNB* classifiers b_i: V_1 **)**
- HashMap H
- **for** *Classifier in* b_i **do** - Assign d to the label class $Class$ having maximum posterior probability
- Add $Class$ count to H
- $majorityClass \leftarrow majority_voting(H)$
- **emit** (*key: d + majorityClass, value: one*)

//ReduceByKey Task 2
reduceByKey(*key: d + majorityClass*, *list*(*ones*) **)**
- HashMap H
- $sum \leftarrow 0$
- **while** *values.hasNext*() **do**
- $sum \leftarrow sum + 1$
- Add "d + majorityClass" count to H
- **close( )**
  - $Max \leftarrow max(H)$
  - $MaxClass \leftarrow Tokenize(key)$
  - $d \leftarrow Tokenize(key)$
  - **emit** (*key: d, value: MaxClass*)

the number of local classifier models at each split. The worker takes an entire input data split as a value and null as a key, then, it performs a data sub-sampling based on a set of available *miki* features over the split. Indeed, thanks to the informativeness of a *miki* over a split of data, our feature selection method based on *miki*s extraction has led to a better accuracy as shown in the next section. Then, for each sub-sample, the worker emits a sub-sample identifier (i.e. *MNB* identifier) as a key and a composed value. The value contains the feature, the class label, the prior probability value, and the feature likelihood (conditional probability of the feature according to its class label). The same worker receives each key (i.e. *MNB* identifier) coupled with its value. Without aggregating the results, the worker just writes the results to the disck.

Job2: Classification

To classify a new unseen instance (i.e. document), in the second Spark job, EEC proceeds by locally loading a set of *MNB* classifiers (generated at its first Spark job) to each worker. Thus, each worker takes a null as a key and a whole set of *MNB* classifiers as a value. Then, the worker reads an unseen instance from a hadoop distributed cache. At each worker, a classifier determines the class label of the unseen instance (i.e. the class label that has the maximum posterior probability). Then, a majority voting scheme is carried out over all classifier results at each worker. Thus, the worker emits the unseen instance as a key coupled with its assigned majority voted class. The emitted value is simply '1', thus all votes go to one worker. Then, the worker aggregates over the received keys and iterates over the values ('1's). At the end of the computation, a close() method is called which is in charge of determining the majority class label. The final result is written to RDD.

6.6 Experiments

To assess the performance of EEC algorithm, we have carried out extensive experimental evaluations. In subsection 6.6.1, we depict our experimental setup, and in subsection 6.6.3, we investigate and discuss the results of our different experiments.

6.6.1 Experimental Setup

We implemented EEC algorithm on the top of Spark, using Java programming language version 1.7 and Spark version 1.6.2. For comparing EEC performance with other text classification algorithms, we adopted the default implementations of existing algorithms in the MLlib machine learning library [110].

We carried out all our experiments by using the NEF [111] platform, which is a platform for large-scale data processing. We have used a cluster of different sizes from 4, to 32 machines. Each machine is equipped with Linux operating system, 64 Gigabytes of main memory, Intel Xeon $X3440$ 4 core CPUs, and 320 Gigabytes SATA hard disk.

In our experiments, we adopt the following notations. We denote by *'EEC'* our proposed method Ensemble of Ensemble of Classifiers, and by *'RandomForest'* the random forest algorithm. Likewise, we denote by *'MultiBayes'* the straightforward parallel implementation of Multinomial Naive Bayes Classifier. We refer by *'SVD'* the parallel singular value decomposition algorithm. Finally, we denote by *'NaiveBayes'* the parallel straightforward implementation of Naive Bayes algorithm based on Bernoulli distribution.

In our different experiments, we evaluate the performance of each algorithm based on its classification accuracy and responding time of execution. At each time, we record the classification accuracy. In particular, we vary one particular parameter, the number of sub-samples stated as features from extracted *miki* over a split of data in each worker with different sizes of k, to better demonstrate our contribution using the *miki* as feature selection method for the learning process.

6.6.2 Datasets

To better evaluate the performance of our EEC algorithm, we performed our experiments based on 2 different real-world datasets. The first dataset is the whole set of the 2014 English Wikipedia articles [112]. This dataset is composed of 5 millions articles and having a size of 49 GB. The second dataset is a sample data of the English ClueWeb dataset [113]. The size of this sample dataset is roughly one terabyte and having 632 million articles.

For English Wikipedia and ClueWeb datasets, we create 500 categories (i.e. class labels). For each dataset, we performed a data cleaning task (we removed all English stop words), where each article represents an instance (i.e. document or example).

6.6.3 Performance Analysis

In this section, we analyze our different experimental results. In particular, we evaluate the classification accuracy performance of each presented method.

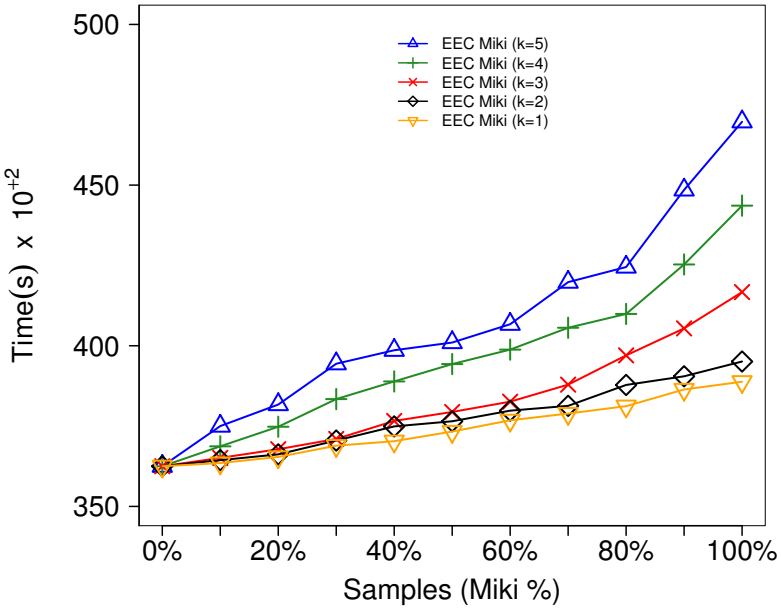


Figure 6.10: Time execution of EEC algorithm over English Wikipeda Articles Dataset varying the quantity of *miki* features.

Figures 6.10 and 6.11 sketch the performances of our EEC algorithm over English Wikipedia dataset and the Clue Web dataset in terms of execution time, varying the percentage of the used features issued from the set of *mikis* as sub-samples. In figure 6.10, we show different versions of our EEC algorithms by varying the size of *mikis* from where we issued the features to proceed our learning process. Obviously, with 100% of features issued from our set of *mikis* as sub-samples, the time response of all the versions of EEC are higher then the other percentage since the additional runtime of the *miki* mining process amplifies the execution time of the hole process. Indeed, an EEC algorithm with features from *mikis* with size $k = 1$ outperforms all the other alternatives, and for the same order of the *mikis* size, EEC versions keep the same order of performances in terms of time execution.

In Figure 6.11, similar experiments have been conducted on the ClueWeb dataset, and we observe very similar behaviors (*i.e.* EEC with feautures issued from *mikis* with size $k = 1$ outperforms the other alternatives, and the same order between all versions of the algorithm is kept).

Tables 6.1 and 6.2 report our experimental results for a better accuracy of our clas-

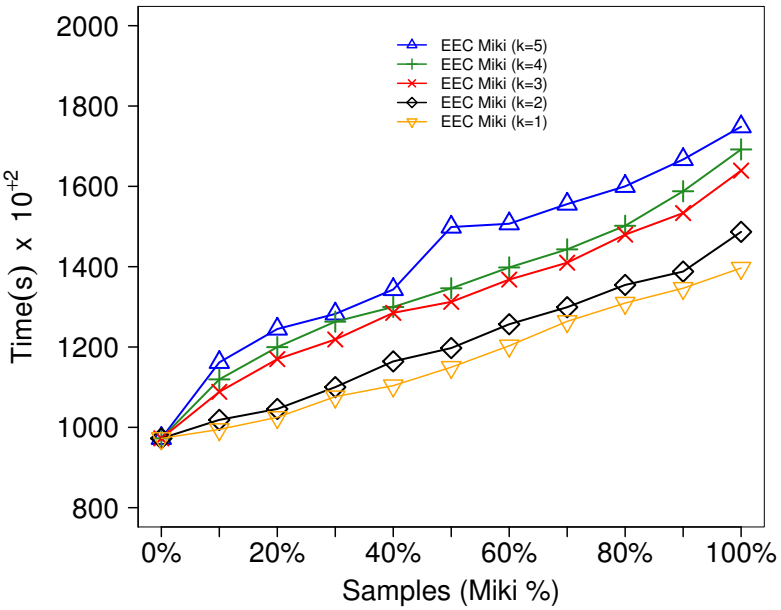


Figure 6.11: Time execution of Eec algorithm over Clue Web Dataset varying the quantity of *miki* features.

sification algorithm on the whole English Wikipedia dataset and the Clue Web dataset, respectively. Table 6.1 shows the accuracy of different versions of Eec versus the number of features as sub-samples varying their quantity from 10% to 100%. In particular, in this experiment, we fixed the number of sub-samples to 500 for every version of Eec. We see that the classification accuracy of Eec with $k <= 3$ is low and the performance of these three algorithms is outperformed by Eec with $k = 4$ and $k = 5$. This normal difference in the classification performance is due to the fact that the features in the text is better modeled by improving the size k of the *miki* in question, which obviously brings more informations about the documents in the split and their assignment to a class is much more accurate. Moreover, Table 6.1 shows a comparison between the Eec algorithm and other alternatives from the literature. We evaluated the accuracies values for Random Forest algorithm and the NaiveBayes algorithm implemented under MlLIB of Spark. We can see a huge difference in the values of accuracies compared to the version of Eec in which we used 100% of features from *mikis* as sub-samples. Even for percentages less then 40%, all Eec versions outperform outstandingly the other alternatives.

Same results of our contribution are exposed in Table 6.2, by sketching the accuracy results of Eec algorithm over Clue web dataset. Indeed, with 40% features from *mikis* in the whole set of the sub-samples,

Table 6.1: Accuracy values on English Wikipedia Articles Dataset

Samples (Miki %)	Random Values (0%)	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Eec k=1	0.873654	0.9052368	0.9112397	0.9123668	0.9257798	0.930659	0.9367571	0.9420058	0.9520463	0.9699549	**0.970075**
Eec k=2	0.873654	0.9165648	0.9188997	0.9200962	0.9279998	0.932998	0.9396321	0.9459698	0.9523656	0.9700351	**0.972612**
Eec k=3	0.873654	0.9187254	0.9200046	0.9311923	0.9399971	0.940982	0.9479215	0.9500954	0.9653253	0.9688452	**0.971866**
Eec k=4	0.873654	0.9200980	0.9256593	0.9378821	0.9455412	0.947765	0.9507985	0.9542000	0.9686289	0.9699995	**0.976501**
Eec k=5	0.873654	0.9236774	0.9278512	0.9399991	0.9499957	0.951299	0.9567986	0.9599810	0.9623423	0.9699999	**0.978876**
Random Forrest	**0.844509**										
NaiveBayes	**0.762893**										

Table 6.2: Accuracy values on Clue Web Dataset

Samples (Miki %)	Random Values (0%)	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Eec k=1	0.792536	0.8800768	0.8900004	0.9019107	0.9034499	0.910653	0.9178298	0.9220463	0.9268549	0.9299998	**0.9300021**
Eec k=2	0.792536	0.8810148	0.8904283	0.9045823	0.9080018	0.912110	0.9199337	0.9256321	0.9299443	0.9335116	**0.9371280**
Eec k=3	0.792536	0.8858645	0.8948546	0.9081113	0.9157008	0.920056	0.9262431	0.9298994	0.9300562	0.9356609	**0.9399929**
Eec k=4	0.792536	0.8878491	0.8988593	0.9158729	0.9199123	0.926129	0.9280946	0.9300080	0.9366128	0.9399854	**0.9400317**
Eec k=5	0.792536	0.8900036	0.9048512	0.9199991	0.9249632	0.928824	0.9288797	0.9330309	0.9347889	0.9399819	**0.9466198**
Random Forrest	**0.714509**										
NaiveBayes	**0.632893**										

6.7 Conclusion

In this chapter, we proposed a reliable and efficient MapReduce based parallel classifier, namely EEC that has shown significant efficiency in terms of runtime and classification accuracy. EEC elegantly achieves a classification task of very large databases by developing a clever principle of two steps decision making technique. This technique results in a very high classification accuracy. EEC takes the full advantage of the available resources in a massively distributed environment to guarantee a low response time in the whole learning process while highly increasing the classification accuracy.

In our experiments, EEC has achieved very good performance when classifying textual data and we believe that the principle and logic behind the working process of EEC algorithm can be used with other standard classification algorithms.

Chapter 7

Conclusion

"Every story has an end, but in life every end is just a new beginning"
Benjamin Franklin

In this chapter, we summarize and discuss the main contributions made in the context of this thesis. Then we give some research directions for future work.

7.1 Contributions

This thesis is in the context of the parallel mining of itemsets in massively distributed environments. We have focused on the itemset discovery problem in big data, aiming to improve and accelerate the itemset discovery processes which are of interest to various applications that deal with big datasets and data streams. In this thesis, we have made two contributions:

7.1.1 Fast Mining of Closed Frequent Itemsets from Big Data

We addressed the problem of mining Closed Frequent Itemsets in massively distributed environments. Our main challenge was to limit the itemset discovery to be done in just one simple, yet very efficient job.

Generally, mining $\mathcal{CFI}$s in more than one job could impact the performance of the

whole mining process. Although our proposal DCIM has achieved very good performance in extracting $\mathcal{CFI}$s in large databases, it accounts for some limitations. The first job of DCIM algorithm could transfer candidate global $\mathcal{CFI}$s (*i.e.* local $\mathcal{CFI}$s) of no use which impacts the whole mining process. In particular, this results in a high data (*miki* local $\mathcal{CFI}$s) transfer between the mappers and the reducers which in turns degrade the performance of the second job. Thus, our main challenge is to omit the second job and perform a $\mathcal{CFI}$ extraction in just one job instead of two. To this end, we proposed in this thesis a second version of DCIM. Based on a clever data transformation technique strategy, DCIM mines each data partition independently. We have extensively evaluated our DCIM algorithm using very large datasets (up to 53 million transaction of data) and very low minimum support, the results confirms the efficiency and effectiveness of our approach.

7.1.2 Fast Parallel Mining of Maximally Informative K-itemset from Data Stream

We addressed the problem of mining *miki* from data streams. Our main challenge was to improve and accelerate the *miki* discovery over batches of incoming and outgoing data. To this end we proposed PENTROS, a highly, scalable algorithm that is capable to extract the *miki* of different sizes in just two simple, yet very efficient jobs. In a massively distributed environment such as Spark, at its first job, PENTROS determines a set of potential *miki* by applying a simple ForwardSelection algorithm at each worker. By using a very clever technique to estimate the joint probabilities of the *miki* candidates at its first job, PENTROS reduces the computations load of its second job. With a bunch of computational optimizations, PENTROS renders the *miki* discovery in very large throughput of data simple and succinct. We evaluate the effectiveness and the capabilities of PENTROS algorithm by carrying out extensive, various experiments with very large real-world data streams. The results have shown an outstanding performance of PHIKS comparing to other alternatives.

7.1.3 Fast Parallel Ensemble of Ensembles of Classifiers

With very large amounts of data, performing a classification task is not easy. This is due to the large numbers of attributes. It is likely that several attributes are of no relevance to the classification task. We developed a new framework intended to improve any classification task by combining two simple, yet very efficient techniques under Spark framework:

The first technique extended the ensemble classifier methods by performing two parallel decision steps in a massively distributed environment. The main idea was to create several simple classifiers (*e.g.* Naive Bayes classifiers) locally say at each worker. This has been achieved by performing a simple sampling (based on the attributes) of the data at each worker. Thus, each worker had a set of say k trained classifiers. To classify a new instance, a test decision is made at each worker. Each worker locally classified the new instance based on a majority voting scheme. The classified instance and its class label are shuffled. In the reduceByKey transformation, each worker received each classified instance

with its class label. The worker further aggregated the results based on the class labels and outputs the final result *i.e.* the classified instance (key) with its majority class (value).

In the second technique we incorporated the *miki* with the two decision steps classification. We used the *miki* to improve the classification task. Since the miki highly discriminate the database, they significantly improved the accuracy of the classification. Instead of using irrelevant attributes in the classification, we first applied our PENTROS algorithm in a static version to extract the relevant attributes and then used them with our two steps decision classification technique.

7.2 Directions for Future Work

With the abundant and various researches that have been carried out to improve the performances of parallel data mining algorithms, one may wonder whether we have solved most of the critical problems related to both frequent and *miki* mining. Particularly, with the challenges that we have been facing with big data, there would be several possible extensions and improvements of the work that we have achieved in this thesis. Some future directions of research are as follows.

- **Using Spark for fast mining of $\mathcal{CFI}$s:** Since Spark [10] supports in-memory computations which is far faster than MapReduce, we highly believe that considering different data placement strategies along with a specific Frequent Itemset Mining ($\mathcal{FIM}$) algorithm in Spark, would improve the global mining process.

  The first job consists of determining a set of local frequent itemsets *i.e.* after applying a specific data placement strategy. Using a flatMap() function, each bunch of the database transactions is loaded into the memory as an RDD (Resilient Distributed dataset) object. Then, a specific FIM algorithm is applied locally on each RDD (partition of the global dataset). The results of the mappers (*i.e.* flatMaps) are emitted to the reducers. A reduceByKey() function is applied to aggregate the different results.

  The second job, consists of validating the global frequency of the local frequent itemsets. The results (local frequent itemsets) are shared between the different workers using an accumulator in Spark. A flatMap() function is applied on the RDDs to count the occurrence of each local frequent itemset in the whole database. Then, a reduceByKey() function is applied to aggregate the results received from the mappers. The reducer sums up the results and outputs the local frequent itemsets that are globally frequent.

  Likewise, as in MapReduce, cleverly partitioning the database transactions allows for a very fast mining of frequent itemsets. Our proposed DCIM algorithm, as described in chapter 4, is outstandingly capable to extract the $\mathcal{CFI}$s from very large databases and with very low minimum support. Based on the Spark framework, we highly

believe that the performance of Dcim algorithm would be very significant. The design and the implementation of our proposed Dcim algorithm in Spark would be very simple. The database partitions are loaded as RDDs to each worker. The worker applies Dcim algorithm locally on its RDD (*i.e.* data partition) and the results are emitted from worker to another.

Bibliography

[1] "John graunt." `http://www.encyclopedia.com/people/medicine/medicine-biographies/john-graunt`. Accessed: 2016-09-30.

[2] "Thomas bayes." `http://www.bookrags.com/biography/thomas-bayes-wom`. Accessed: 2017-02-31.

[3] "Rakesh agrawal." `http://rakeshagrawal.org/`. Accessed: 2016-09-30.

[4] L. Feng, Y. K. Chiam, and S. K. Lo, "Text-mining techniques and tools for systematic literature reviews: A systematic literature review," in *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, pp. 41–50, 2017.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of 20th International Conference on Very Large Data Bases, VLDB'94*, (Santiago de Chile, Chile), pp. 487–499, 1994.

[6] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts," in *Ordered Sets*, (Dordrecht, Boston, USA), pp. 445–470, 1982.

[7] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, pp. 398–416, 1999.

[8] S. Li, L. Li, and C. Han, "Mining closed frequent itemset based on fp-tree," in *The 2009 IEEE International Conference on Granular Computing, GrC 2009, Lushan Mountain, Nanchang, China, 17-19 August 2009*, pp. 354–357, 2009.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Journal of Commun. ACM*, pp. 107–113, 2008.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *HotCloud 2010*, (Boston, USA), 2010.

[11] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 6th International Conference on Management of Data, SIGMOD'00*, (Dallas, Texas, USA), pp. 1–12, 2000.

[12] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: parallel fp-growth for query recommendation," in *Proceedings of the 2nd International Conference on Recommender Systems*, (Lausanne, Switzerland), pp. 107–114, 2008.

[13] A. J. Knobbe and E. K. Y. Ho, "Maximally informative k-itemsets and their efficient discovery," in *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining, ACMSIGKDD'06*, (Philadelphia, PA, USA), pp. 237–244, 2006.

[14] N. V. Sawant, K. Shah, and V. A. Bharadi, "Survey on data mining classification techniques," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, (New York, USA), pp. 1380–1420, 2011.

[15] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pp. 432–444, 1995.

[16] W. Song, B. Yang, and Z. Xu, "Index-bittablefi: An improved algorithm for mining frequent itemsets," *Journal of Knowl. Based Syst.*, pp. 507–513, 2008.

[17] N. Jayalakshmi, V. Vidhya, M. Krishnamurthy, and A. Kannan, "Frequent itemset generation using double hashing technique," *Journal of Procedia Engineering*, pp. 1467 – 1478.

[18] M. Zaki, "Scalable algorithms for association mining," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, pp. 372–390, May 2000.

[19] Y.-J. Tsay and Y.-W. Chang-Chien, "An efficient cluster and decomposition algorithm for mining association rules," *Inf. Sci*, vol. 160, no. 1-4, pp. 161–171, 2004.

[20] N. Pasquier, "Datamining: Algorithmes d'extraction et de réduction des règles d'association dans les bases de données," thèse de doctorat, Ecole Doctorale Sciences pour l'Ingénieur de Clermont Ferrand, Université Clermont Ferrand II, France, Janvier 2000.

[21] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[22] S. B. Yahia and E. M. Nguifo, "Approches d'extraction de règles d'association basées sur la correspondance de galois," *Ingénierie des Systèmes d'Information*, pp. 23–55, 2004.

[23] A. J. Knobbe and E. K. Y. Ho, "Maximally informative k-itemsets and their efficient discovery," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 237–244, 2006.

[24] C. Nevill-Manning, G. Holmes, and I. H. Witten, "The development of holte's 1r classifier," in *Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference on*, pp. 239–242, Nov 1995.

[25] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[26] S. Ruggieri, "Efficient c4.5 [classification algorithm]," *Knowledge and Data Engineering, IEEE Transactions on*, pp. 438–444, 2002.

[27] D. Lowd and P. Domingos, "Naive bayes models for probability estimation," in *Proceedings of the 22Nd International Conference on Machine Learning, ICML '05*, (New York, USA), pp. 529–536, 2005.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, pp. 273–297.

[29] W. W. Cohen and Y. Singer, "Context-sensitive learning methods for text categorization," *ACM Trans. Inf. Syst.*, vol. 17, pp. 141–173, Apr. 1999.

[30] C. Aggarwal and C. Zhai, "A survey of text classification algorithms," in *Mining Text Data*, pp. 163–222, 2012.

[31] D. Landgrebe, "A survey of decision tree classifier methodology," *Systems, Man and Cybernetics, IEEE Transactions on*, pp. 660–674, 1991.

[32] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," in *Machine Learning and Its Applications, Advanced Lectures*, pp. 249–257, 2001.

[33] V. Cherkassky, "The nature of statistical learning theory," *IEEE Trans. Neural Networks*, vol. 8, no. 6, p. 1564, 1997.

[34] S. Raschka, "Naive bayes and text classification I - introduction and theory," *CoRR*, vol. abs/1410.5329, 2014.

[35] M. Sewell, "Ensemble learning," 2011.

[36] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[37] P. Komarek, *Logistic Regression for Data Mining and High-Dimensional Classification*. PhD thesis, Pittsburgh, PA, May 2004.

[38] L. Mitchell, T. M. Sloan, M. Mewissen, P. Ghazal, T. Forster, M. Piotrowski, and A. S. Trew, "A parallel random forest classifier for r," in *Proceedings of the Second International Workshop on Emerging Computational Methods for the Life Sciences*, (New York, USA).

[39] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[40] M. Berry, *Survey of Text Mining Clustering, Classification, and Retrieval.* New York, NY: Springer New York, 2004.

[41] Z. Qin, *Introduction to e-commerce.* Berlin London: Springer, 2009.

[42] C. Loglisci, M. Berardi, S. D'Alessandro, and P. Leo, "Finding generalized closed frequent itemsets for mining non redundant association rules," in *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007, 17-20 June 2007, Torre Canne, Fasano, BR, Italy*, pp. 184–195, 2007.

[43] A. Casali, R. Cicchetti, and L. Lakhal, "Essential patterns: A perfect cover of frequent patterns," in *Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery, DaWaK'05*, (Copenhagen, Denmark), pp. 428–437, 2005.

[44] G. Liu, J. Li, and L. Wong, "A new concise representation of frequent itemsets using generators and a positive border," *Knowl. Inf. Syst.*, vol. 17, no. 1, pp. 35–56, 2008.

[45] T. Hamrouni, S. B. Yahia, and E. M. Nguifo, "Sweeping the disjunctive search space towards mining new exact concise representations of frequent itemsets," *Data Knowl. Eng.*, vol. 68, no. 10, pp. 1091–1111, 2009.

[46] Han, Pei, and Yin, "Mining frequent patterns without candidate generation," *SIGMODREC: ACM SIGMOD Record*, vol. 29, 2000.

[47] I. S. Janos Galambos, *Bonferroni-type Inequalities with Applications.* New York, USA: Springer, 1996.

[48] F. Bonchi and C. Lucchese, "On condensed representations of constrained frequent patterns," *Knowl. Inf. Syst.*, vol. 9, no. 2, pp. 180–201, 2006.

[49] C. Aggarwal, *Frequent pattern mining.* Cham: Springer, 2014.

[50] B. Ganter, G. Stumme, and R. Wille, eds., *Formal Concept Analysis, Foundations and Applications*, vol. 3626 of *Lecture Notes in Computer Science*, Springer, 2005.

[51] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," *Data Min. Knowl. Discov.*, vol. 1, no. 3, pp. 241–258, 1997.

[52] J. Boulicaut, L. D. Raedt, and H. Mannila, eds., *Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers*, vol. 3848 of *Lecture Notes in Computer Science*, Springer, 2005.

[53] D. Serrano and C. Antunes, "Condensed representation of frequent itemsets," in *18th International Database Engineering & Applications Symposium, IDEAS 2014, Porto, Portugal, July 7-9, 2014*, pp. 168–175, 2014.

[54] E. Simoudis, J. Han, and U. M. Fayyad, eds., *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, AAAI Press, 1996.

[55] Z. Liu, L. Hu, C. Wu, Y. Ding, Q. Wen, and J. Zhao, "A novel process-based association rule approach through maximal frequent itemsets for big data processing," *Future Generation Comp. Syst.*, pp. 414–424, 2018.

[56] M. L. Wong, W. Lam, and K. Leung, "Using evolutionary programming and minimum description length principle for data mining of bayesian networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 2, pp. 174–178, 1999.

[57] J. Baixeries, C. Sacarea, and M. Ojeda-Aciego, eds., *Formal Concept Analysis - 13th International Conference, ICFCA 2015, Nerja, Spain, June 23-26, 2015, Proceedings*, vol. 9113 of *Lecture Notes in Computer Science*, Springer, 2015.

[58] H. Liu, L. Liu, and H. Zhang, "A fast pruning redundant rule method using galois connection," *Appl. Soft Comput.*, pp. 130–137, 2011.

[59] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Advances in knowledge discovery and data mining," ch. From Data Mining to Knowledge Discovery: An Overview, pp. 1–34, Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996.

[60] S. BenYahia, T. Hamrouni, and E. M. Nguifo, "Frequent closed itemset based algorithms: a through structural and analytical survey," *Journal of SIGKDD Explorations*, pp. 93–104, 2006.

[61] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Inf. Syst.*, vol. 24, no. 1, pp. 25–46, 1999.

[62] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal, "Computing iceberg concept lattices with Titanic," *Journal of Data Knowledge Engineering*, pp. 189–222, 2002.

[63] Y. Wang, Y. Jin, Y. Li, and K. Geng, "DM data mining based on improved apriori algorithm," in *Information Computing and Applications - 4th International Conference, ICICA 2013, Singapore, August 16-18, 2013 Revised Selected Papers, Part II*, pp. 354–363, 2013.

[64] J. Pei, J. Han, and R. Mao, "CLOSET: an efficient algorithm for mining frequent closed itemsets," in *Workshop on Research Issues in Data Mining and Knowledge Discovery ACM SIGMOD'00*, pp. 21–30, 2000.

[65] J. Wang, J. Han, and J. Pei, "CLOSET+: searching for the best strategies for mining frequent closed itemsets," in *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining, KDD'03*, (Washington, DC, USA), pp. 236–245, 2003.

[66] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," in *Proceedings of the 3rd Workshop on Frequent Itemset Mining Implementations, ICDM'03*, (Melbourne, Florida, USA), pp. 249–274, 2003.

[67] M. J. Zaki and C. Hsiao, "CHARM: an efficient algorithm for closed itemset mining," in *Proceedings of the 2nd International Conference on Data Mining SIAM'02*, (Arlington, VA, USA), pp. 457–473, 2002.

[68] C. Lucchese, S. Orlando, and R. Perego, "DCI closed: A fast and memory efficient algorithm to mine frequent closed itemsets," in *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, 2004.

[69] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "LCM: an efficient algorithm for enumerating frequent closed item sets," in *Proceedings of The 3rd IEEE International Conference on Data Mining, ICDM'03*, (Melbourne, Florida, USA), 2003.

[70] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *Journal of IEEE Transactions on Knowledge and Data Engineering*, pp. 21–36, 2006.

[71] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "An efficient algorithm for enumerating closed patterns in transaction databases," in *Proceedings of the 7th International Conference on Discovery Science, DS'04*, (Padova, Italy).

[72] T. M. Cover, *Elements of information theory*. Hoboken, N.J: Wiley-Interscience, 2006.

[73] H. Heikinheimo, E. Hinkkanen, H. Mannila, T. Mielikäinen, and J. K. Seppänen, "Finding low-entropy sets and trees from binary data," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 350–359, 2007.

[74] S. Brin, R. Motwani, and C. Silverstein, "Beyond market baskets: Generalizing association rules to correlations," *SIGMOD Rec.*, vol. 26, pp. 265–276, June 1997.

[75] N. Tatti, "Probably the best itemsets," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pp. 293–302, 2010.

[76] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining frequent patterns in data streams at multiple time granularities," 2002.

[77] W.-G. Teng, M.-S. Chen, and P. S. Yu, "A regression-based temporal pattern mining scheme for data streams," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pp. 93–104, 2003.

[78] C. Zhang and F. Masseglia, "Discovering highly informative feature sets from data streams," in *Database and Expert Systems Applications*, pp. 91–104, 2010.

[79] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers and Electrical Engineering*, vol. 40, no. 1, pp. 16 – 28, 2014.

[80] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, mar 2003.

[81] "Hadoop." http://hadoop.apache.org.

[82] "mesos." http://mesos.apache.org.

[83] "Casandra." http://cassandra.apache.org.

[84] "https://www.datanami.com/2015/11/30/spark-streaming-what-is-it-and-whos-using-it/."

[85] R. Anand, *Mining of massive datasets.* 2012.

[86] C. Lucchese, S. Orlando, and R. Perego, "Parallel mining of frequent closed patterns: Harnessing modern computer architectures," in *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pp. 242–251, 2007.

[87] S. Wang, Y. Yang, Y. Gao, G. Chen, and Y. Zhang, "Mapreduce-based closed frequent itemset mining with efficient redundancy filtering," in *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012*, pp. 449–453, 2012.

[88] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in *Proceedings of the 1st IEEE International Conference on Big Data*, (Santa Clara, CA, USA), pp. 111–118, 2013.

[89] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer, "Event log mining tool for large scale hpc systems," in *Proceedings of Euro-Par'11*, (Berlin, Heidelberg), 2011.

[90] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Mining console logs for large-scale system problem detection," in *Proceedings of SysML'08*, (Berkeley, CA, USA), 2008.

[91] K. Chen, L. Zhang, S. Li, and W. Ke, "Research on association rules parallel algorithm based on fp-growth," in *Proceedings of the International Conference on Information Computing and Applications, ICICA'11*, (Qinhuangdao, China), pp. 249–256, 2011.

[92] S. Wang, Y. Yang, Y. Gao, G. Chen, and Y. Zhang, "Mapreduce-based closed frequent itemset mining with efficient redundancy filtering," in *Proceedings of the 12th IEEE International Conference on Data Mining*, (Brussels, Belgium), pp. 449–453, 2012.

[93] S. Wang and L. Wang, "An implementation of fp-growth algorithm based on high level data structures of weka-jung framework," *Journal of Cases on Information Technology*, pp. 287–294, 2010.

[94] C. Nègre, "Efficient binary polynomial multiplication based on optimized karatsuba reconstruction," *Journal of Cryptographic Engineering*, pp. 91–106, 2014.

[95] A. Zanoni, "Iterative toom-cook methods for very unbalanced long integer multiplication," in *Proceedings of the 35th International Symposium in Symbolic and Algebraic Computation, ISSAC'10*, (Munich, Germany), pp. 319–323, 2010.

[96] F. Gorunescu, *Data Mining - Concepts, Models and Techniques.* Springer, 2011.

[97] S. Salah, R. Akbarinia, and F. Masseglia, "Fast parallel mining of maximally informative k-itemsets in big data," in *Proceedings of the 2015 IEEE International Conference on Data Mining ICDM*, (Atlantic City, NJ, USA), pp. 359–368, 2015.

[98] T. A. Runkler, *Data Analytics - Models and Algorithms for Intelligent Data Analysis.* Springer, 2016.

[99] Y. Bassil, "A survey on information retrieval, text categorization, and web crawling," *CoRR*, 2012.

[100] T. M. Cover and J. A. Thomas, *Elements of information theory (2. ed.).* Wiley, 2006.

[101] C. Zhang and F. Masseglia, "Discovering highly informative feature sets from data streams," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications, DEXA'10*, (Bilbao, Spain), pp. 91–104, 2010.

[102] O. Papapetrou, M. N. Garofalakis, and A. Deligiannakis, "Sketching distributed sliding-window data streams," *The VLDB Journal*, 2015.

[103] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2010.

[104] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, pp. 21–27, 2006.

[105] F. Sebastiani, "Machine learning in automated text categorization," *Journal of ACM Comput. Surv.*, pp. 1–47.

[106] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," 1998.

[107] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *In Proceedings of the Twentieth International Conference on Machine Learning*, pp. 616–623, 2003.

[108] J. Tang, S. Alelyani, and H. Liu, "Feature selection for classification: A review," in *Data Classification: Algorithms and Applications*, pp. 37–64, 2014.

[109] "Trec." `http://plg.uwaterloo.ca/~gvcormac/treccorpus/`.

[110] S. Owen, *Mahout in action*. Shelter Island, N.Y: Manning Publications Co, 2012.

[111] "Nef cluster." `https://wiki.inria.fr/ClustersSophia/Clusters_Home`, 2015.

[112] "English wikipedia articles." `http://dumps.wikimedia.org/enwiki/latest`, 2014.

[113] "The clueweb09 dataset." `http://www.lemurproject.org/clueweb09.php/`, 2009.