Beyond Itemsets: Mining Frequent Featuresets over Structured Items

Saravanan Thirumuruganathan[†], Habibur Rahman[†], Sofiane Abbar[‡], Gautam Das[†]
[†]University of Texas at Arlington, [‡]Qatar Computing Research Institute
[†]{saravanan.thirumuruganathan@mavs, habibur.rahman@mavs, gdas@cse}.uta.edu
[‡]sabbar@qf.org.qa

ABSTRACT

We assume a dataset of transactions generated by a set of users over structured items where each item could be described through a set of features. In this paper, we are interested in identifying the frequent featuresets (set of features) by mining item transactions. For example, in a news website, items correspond to news articles, the features are the named-entities/topics in the articles and an item transaction would be the set of news articles read by a user within the same session. We show that mining frequent featuresets over structured item transactions is a novel problem and show that straightforward extensions of existing frequent itemset mining techniques provide unsatisfactory results. This is due to the fact that while users are drawn to each item in the transaction due to a subset of its features, the transaction by itself does not provide any information about such underlying preferred features of users. In order to overcome this hurdle, we propose a featureset uncertainty model where each item transaction could have been generated by various featuresets with different probabilities. We describe a novel approach to transform item transactions into uncertain transaction over featuresets and estimate their probabilities using constrained least squares based approach. We propose diverse algorithms to mine frequent featuresets. Our experimental evaluation provides a comparative analysis of the different approaches proposed.

1. INTRODUCTION

1.1 Frequent Featureset Mining Problem

Technological progress has now enabled businesses to collect fine-grained information about how users interact with their websites and applications. Such information can include the articles read, movies watched, items purchased etc. We consider a database of *structured items* where each item can be described through a set of *features*. Intuitively, we can represent each feature as a boolean attribute whose presence or absence in an item can be observed. By interacting with such items, users generate *item transactions* (transactions over items.) For example, the set of articles read by the user on any given day forms a transaction.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 3
Copyright 2014 VLDB Endowment 2150-8097/14/11.

Frequent itemset mining is an important first step in data analysis for a broad class of applications. It returns a collection of itemsets (set of items) that are most commonly consumed together. A huge body of research literature studies this problem under various scenarios. In contrast to prior work that tries to identify *itemsets* from item transactions (under various conditions), we consider a novel problem of discovering frequent *featuresets* (set of features) from item transactions.

> FREQUENT FEATURESETS MINING (FFM) PROBLEM: Given an database containing transactions over structured items, identify the set of frequent featuresets.

The frequent featureset mining problem has variety of applications and is general enough to handle any scenario where transactions over structured items are performed.

- Consider a music website such as *last.fm*. Here the items correspond to songs and the features are the artists associated with the song. The set of songs listened in a session can be treated as item transaction. FFM tries to identify the artists whose songs are listened together (without user mentioning why she listens to a specific song).
- Consider a news website such as *New York Times*. Here the items correspond to news articles, the features are the named entities (such as Obama or Egypt) in the article, while a transaction could be the set of articles read by a user in a session. Instead of identifying the most commonly read articles (as done in traditional itemset mining), FFM tries to identify the *features* that are consumed together.
- Consider a movie website such as *IMDB*. The movies correspond to items while the features could be actors, directors, genre etc. The set of movies watched (or rated) by a user in a given period time could be considered as a transaction. FFM tries to identify the set of features (such as an actor, director combination) that are consumed together.

Problem Novelty: In contrast to traditional itemset mining, we consider items that are structured with rich features. Further, while the user interaction with the item could be observed, the corresponding user interaction with features is typically hidden. For example, while *New York Times* knows *which* articles the user read, it does not know *why* (what features in the article led the user to read them). Our objective in this paper is to identify frequent patterns over these "hidden" interactions. Notice that identifying the set of articles most frequently read belong to the traditional itemset mining problem. However, using the entire article transaction log to identify the user's favorite topic is a featureset mining problem. This problem is quite challenging as we have to identify the most *important* feature without any explicit feedback from the user.

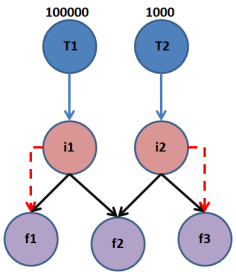


Figure 1: Hierarchical Representation of Transactions over Structured Items

Consider Figure 1 that represents a small dataset of 2 item transactions $\{T_1, T_2\}$. For simplicity, we assume that each transaction contains only one item and that the user consumes an item due to a single feature. The frequency of the transaction is specified above it. T_1 has been observed 100,000 times while T_2 was observed 1000 times. The dataset has two items $\{i_1, i_2\}$ and three features $\{f_1, f_2, f_3\}$. The set of items in a transaction and the set of features present in an item are both observable. The blue arrows connect the transactions to the set of items while black arrows connect structured items to their features. The red dotted arrows highlight the feature which caused the user to consume the corresponding item. For example, the user consumed i_1 due to f_1. While the solid arrows are observable, the dotted ones are not. Hence the major challenge of FFM is to identify that f_1 is a frequent feature instead of f_2 without any explicit feedback from user.

1.2 Challenges

We now consider the various challenges in the general FFM problem where, an item could be consumed by a user due to any arbitrary subset of its features. Further, the rationale for different users to consume the same item could be very different. For instance, on *New York Times*, one user may read an Sports section article due to of its coverage of games while another user may read the same article for its coverage of the players involved. Of course, user transactions on *New York Times* do not provide any information about why an item was consumed. This hidden mapping between items and features makes identifying frequent featuresets extremely challenging.

It might seem that running an existing frequent itemset mining algorithm by aggregating the features could identify frequent featuresets. For example, item i_1 in transaction T_1 is replaced $\{f_1, f_2\}$ while i_2 in T_2 is replaced by $\{f_2, f_3\}$. This approach, while intuitive, results in potentially incorrect frequent featureset due to a subtle pitfall - *the most frequent feature is not necessarily the most important one*.

Frequency Vs Importance. Using the dataset from Figure 1, this results in f_2 being the most frequent featureset followed by f_1. However, we can make a simple argument to negate this conclusion. If feature f_2 was indeed the dominant feature, then more users would have also consumed item i_2. Instead, it languishes with only 1000 transactions. Hence it is clear that the feature f_1 must have a higher importance than f_2 and must have been declared the most frequent featureset. The root cause for the incorrect conclusion is due to the fact that traditional itemset mining approaches considers *all* the features of the items in transaction instead of considering only the subset for which the user picked those items. We highlight other pitfalls of straightforward adaptations of existing deterministic and probabilistic frequent itemset mining algorithms in Section 4.

Combinatorial Explosion. Another major problem is that of *combinatorial explosion* of potential featuresets that could have generated the item transaction database. Consider for instance a single transaction of ten items with 5 features each. If the features of items in the transaction do not overlap, then are there are 50 different features. Even if we assumed that a user picked an item for a single feature, there are 5^{10} potential featuresets that could have generated this item transaction. If we allow the user to choose an item over any subset of its features, then there are exactly 2^5 feature combinations to pick it. Thus, there are $\left(2^5\right)^{10} = 2^{50}$ hidden feature transactions that would generated the single item transaction.

1.3 Outline of Our Approach

Intuitively, any deterministic approach could not be used to solve the FFM problem. Unlike deterministic frequent itemset mining where the presence of an item in a transaction is known with certainty, it is hard to say which subset of features of that item lead the user to pick that item in the transaction. The difficulty comes from the fact that the presence of a feature in a transaction depends on whether the user was interested in that feature while generating the transaction rather than on the presence of that feature in one of the items of the transaction.

We tackle this problem in two stages. First, we identify the potential reasons (featuresets) for which an item was consumed in the context of the transaction/user. Due to the limited user-item interaction information, it is unlikely that we could identify the featureset that generated the transaction with any certainty. Given an item transaction T, we enumerate the various featuresets that could have generated it. We introduce a novel *featureset uncertainty* model to represent the likelihood for each of the candidate featuresets to have generated T. Identifying this likelihood is the *first* fundamental problem we solve in this paper. We use constrained least squares based approach to estimate the likelihood. Our *second* problem seeks to mine the frequent featuresets under the featureset uncertain model. We propose an efficient dynamic programming based approach for this purpose. In an effort to improve the performance, we also designed a scalable "approximation" algorithm with only a marginal decrease in accuracy.

Our experimental results over a number of large datasets show that the frequent featuresets identified by our algorithms have high qualitative score under a number of popular interestingness measures. Additionally, in contrast to traditional itemset mining, the availability of user information allows us to perform personalized featureset mining.

Summary of Contributions.

- We introduce and motivate the novel problem of *frequent featureset mining* for transactions over structured items.
- We describe various approaches to map item transactions to feature transactions and highlight their pros and cons. We also make observations about the subtle issues that render adaptations of traditional itemset mining inapplicable.
- We introduce a novel featureset uncertainty model and develop an efficient algorithm to estimate the likelihood that a featureset generated an item transaction.
- We develop diverse algorithms to mine frequent featuresets under featureset uncertainty model .
- We present a thorough experimental evaluation of our algorithms and study their scalability using large synthetic datasets.

2. FRAMEWORK AND PROBLEM DEFINITION

In this section, we define the data model and the necessary background. We formally state the two central problems addressed in the paper - estimating the likelihood that a featureset generated an item transaction and using this information to mine frequent featuresets. We then describe two variants that occur in practice depending on whether the user who made the transactions is identifiable.

2.1 Framework

Structured Items and Features. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a universe of structured items. Each item $I \in \mathcal{I}$ could be described as a set of features. A *feature* is a property of the item whose presence or absence can be observed. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ be the universe of all features. An item can be equivalently considered as a tuple over boolean attributes $\{f_1, f_2, \ldots, f_m\}$. Given an arbitrary item I and feature f, $I[f] = 1$ if I contains f and 0 otherwise. We represent the set of features of an item I by $features(I)$.

As an example, consider an online news website such as New York Times. Each news article can be considered as a structured item. $\mathcal{I}$ is the entire news article catalog. The set of named entities present in an article can be considered as its features. $\mathcal{F}$ is the set of all the named entities covered by the catalog.

A non empty set of items, $X \subseteq \mathcal{I}$ is called an *itemset*. An itemset is called as *l-itemset* if it has exactly l items. Similarly, a *featureset* $F \subseteq \mathcal{F}$ is a non empty set of features. A *l-featureset* has exactly l features. An item I is said to contain a featureset F, iff $\forall f \in F, I[f] = 1$. We will henceforth use the word *item* to refer to structured item.

Item Transactions. Let $\mathcal{IT} = \{T_1, T_2, \ldots, T_N\}$ be a database of item transactions, $|\mathcal{IT}|$=N. We represent each transaction as a triple $< tid, uid, X >$, where *tid* is the transaction identifier, *uid* is the user identifier who made the transaction and $X \subseteq \mathcal{I}$ is the set of items in the transaction. A transaction is said to contain an itemset if all the items in the itemset are also present in the transaction. Further, the notation $features(T)$ for a transaction T returns the set of features that are present in at least one item in T.

User-Transaction Interaction Information. We consider the database of item transactions $\mathcal{IT}$ to be available at two levels of granularity - aggregate and fine-grained. Under the *aggregate* granularity, the only information available are the transaction identifier and the items contained in the transaction. Under the *fine-grained* granularity, we also know the user id that identifies the user who made the transaction. This additional information allows us to group the transactions by users. We hasten to add that our algorithms do not require any additional profile information about the user. Under aggregate granularity, we would consider all the items to be performed by a single "average" user and use the same user identifier for all item transactions.

Such interaction can be represented via an *aggregate interaction vector* v where each component corresponds to the number of times a particular transaction was made. If the user who made the transaction is known, then we could compute, for each user u, an *individual interaction vector* v_u where each component provides the number of times u performed a given transaction. The interaction vector is normalized and has non-negative numbers that add upto 1.

Frequent Itemset Mining. The support of an itemset X, denoted by $sup(X)$, is the number of transactions in which X appears in the transaction database. Let *minSup* $\in (0, N]$ be an integer, where N is the number of transactions. An itemset X is considered frequent if $sup(X) > minSup$.

Typically, items occurring in a transaction are considered to be certain. In other words, an item exists in a transaction or it doesn't. However, there are scenarios where the presence of an item is uncertain and quantified probabilistically [2, 6]. For example, in our paper, we observe only the item transaction created by the user. However, the underlying featureset that generated the item transaction is typically unknown. A natural model to manage such uncertainty is to assign values to various featuresets based on the likelihood that they have created the item transaction.

Uncertain Transaction Databases. An *uncertain* item I [2] is one whose presence in an item transaction T is provided by its *existential probability* $P(I \in T) \in [0, 1]$. In contrast, a *certain* item either occurs or does not in a transaction. i.e. $P(I \in T) \in \{0, 1\}$. An *uncertain transaction* contains uncertain items. Finally, an *uncertain transaction database* [6] contains uncertain transactions.

Probabilistic Frequent Itemset Mining. We can see that the support of an itemset X, $sup(X)$ is a random variable in uncertain databases and no longer has a constant value. There are multiple candidate definitions that extend support of an itemset to uncertain scenario [9, 7]. However, [20] showed the two most popular definitions (based on expected support and frequent probability) have a tight correlation between them. Given an uncertain database, a minimum support *minSup* and an itemset X, the *frequent probability* of X is defined as:

$$Pr(X) = Pr\{sup(X) \geq minSup\} \quad (1)$$

X is considered to be a *probabilistic frequent itemset* if $Pr(X) \geq minSup$ is at least a constant probabilistic frequent threshold of pft .

Running Example. Table 1 describes a simple dataset with $n = 3$ items, $m = 4$ features and $N = 3$ transactions that will serve as a running example to highlight our various algorithms.

Table 1: Running Example

TId	Item {features}	Count
T_1	$I_1\{f_1, f_3\}, I_2\{f_3, f_4\}$	50
T_2	$I_3\{f_2, f_3\}, I_2\{f_3, f_4\}$	100
T_3	$I_1\{f_1, f_3\}, I_2\{f_3, f_4\}, I_3\{f_2, f_3\}$	1000

2.2 Featureset Uncertainty Model

Model for Generating Item Transactions We first present an intuitive generative model for item transaction using their component features. As we argued in the introduction, a user picks an item due to a subset of its features. Such a behavior extends for each item in a transaction. To generate an item transaction T, the user first picks the size of the transaction, $|T|$. Each item in transaction is chosen as follows : the user first picks a featureset $F \subseteq \mathcal{F}$ according to a probability distribution. Then among the items that contain all the features in featureset F, she selects an item uniformly at random. This process is repeated $|T|$ times to generate the item transaction.

Generating Featureset. Recall that each item in a transaction was chosen by the user due to a subset of its features. Given a transaction $T = \{I_1, I_2, \ldots, I_{|T|}\}$, let F_i be the subset of features for which item I_i was chosen. We refer to the union of these featuresets $G_T = \cup_{i=1}^{|T|} F_i$ as the generating featureset for the item transaction T. In other words, G_T is the "ground-truth" that generated T. Ideally, if our aim is to identify the frequent featuresets, the frequent

mining algorithm must be run over G_T for each item transaction $T \in \mathcal{IT}$.

Featureset Uncertainty Model. However, in practice, it is unlikely that we will be able to ascertain the featureset that generated a transaction deterministically. Further, for any given item transaction T, there are numerous possible featuresets that could have generated T. The set of all featuresets that could have generated T is given by, $\mathcal{G}_T = \{G_T | G_T \subseteq \mathcal{F} \wedge |G_T \cap I| > 0 \; \forall I \in T\}$.

Given a single transaction T and no additional information, we cannot ascertain which of the featureset $F_T \in \mathcal{G}_T$ could have generated T. In our paper, we associate with each featureset the probability that it could have generated the transaction given other item transactions. Of course, given a single item transaction T, each generating featureset in $\mathcal{G}_T$ has a uniform probability (as we do not possess adequate information to claim otherwise). However, given multiple item transactions, it is possible to assign different likelihoods to each generating featureset. Hence, frequent featuresets can be obtained by running uncertain frequent mining algorithms over $\mathcal{FT}$.

Featureset Likelihood. The likelihood of a featureset F for a given transaction T is defined as the conditional probability that F was the generating featureset for T given the entire item transaction dataset $\mathcal{I}$ and all possible generating featureset G_T for T. The featureset likelihood of transaction is a probability distribution over all featuresets in its generating featureset. Of course, any featureset not in $\mathcal{G}_T$ will have a likelihood of 0. We would like to note that the likelihood of a featureset formalizes the notion of importance first described in Introduction.

PROBLEM 1 (**Featureset Likelihood Estimation: FLE**). Given an item transaction database $\mathcal{IT}$ containing user transactions over structured items and aggregate interaction vector v, estimate the featureset likelihood for each transaction $T \in \mathcal{IT}$.

2.3 Frequent Featureset Mining

Once the relative likelihood of the generating featuresets of an item transaction are identified, the next step is to use this information to mine frequent featuresets.

PROBLEM 2 (**Frequent Featureset Mining : FFM**). Given an item transaction database $\mathcal{IT}$ containing user transactions over structured items, the corresponding featureset likelihood and minimum support threshold $minSup$, identify the set of frequent featuresets with expected support of at least $minSup$.

3. OVERVIEW OF OUR APPROACH

In this subsection, we describe the high level components of our approach that also provide a roadmap to the rest of the technical sections.

Challenge I: Enumerating Generating Featuresets. The first major challenge is to generate all possible featuresets that could have generated a given item transaction. However, even for small item transaction, this number could be exponential. Hence we make a simplifying assumption by noting that most users consume an item due to a small subset of its features. Recall that each item in the transaction must have at least one feature in the generating featureset. If we consider each item as a set with its feature as the elements, then a candidate generating featureset is a *hitting set* (a set of elements that has non empty intersection with all the other sets [12]). Section 4 talks about generating featuresets in more detail.

Challenge II: Identifying Likelihood of Generating Featuresets. Once we have identified the generating featuresets for each item transaction in the dataset, we represent them as a bipartite graph where one partition corresponds to item transactions while the other partition corresponds to featuresets. An edge exists between a transaction T and a featureset F if F was a generating featureset for T. Given this bipartite graph, the next challenge is to identify the relative likelihood of each featureset to generate the item transaction. We treat this as a constrained optimization problem, the solution of which allows us to order the generating featuresets of an item transaction based on the likelihood that it generated the transaction. Section 4 talks about identifying the likelihood of featuresets in more detail.

Challenge III: Mining Frequent Featuresets. Once the relative likelihood of the generating featuresets of an item transaction are identified, the next step is to identify the frequent featuresets. We first describe an exact but inefficient algorithm followed a much more efficient but approximate variant. Section 5 talks about mining featuresets in more detail.

Challenge IV: Scaling Featureset Mining. Numerous hurdles exist in each of the prior stages that hinder our ability to develop scalable solution for frequent featureset mining. We utilize sampling as the key technique to achieve scalability. We propose two different algorithms based on sampling the item transaction database and sampling the item transaction-featureset bipartite graph that provide a nice tradeoff between performance and quality . Section 6 talks about our scalability approaches.

4. BASELINE TECHNIQUES

In this section, we describe two intuitive baseline ideas for solving the frequent featureset mining problem and point out their respective pitfalls which motivate our proposed approach. Both techniques solve the problem by transforming the item transaction database into a feature transaction database and utilize existing itemset mining algorithms.

4.1 Adapting Frequent Itemset Mining Algorithms.

In this subsection, we describe two approaches that transform the *certain* item transaction database into another *certain* feature transaction database.

Consider an obvious approach that transforms item transactions into transactions over features by using the union of features present in all the items in the transaction. This approach could lead to inaccurate results as it ignores two important facts: (a) The frequency of features within a transaction. (b) The combination of features for which a user picked the item. If multiple items in a given transaction contain the same feature (for example, multiple articles read by a user contains the topic *Egypt*), then it is highly likely that this feature must be given a higher weight. Taking the union of features fails in capturing this valuable information. Second, this approach considers *all* the features of the items in transaction instead of considering only the subset for which the user picked those items.

However, even assigning a weight based on its frequency does not solve the issue as we show below. Consider a more sophisticated approach where each item transaction T is converted to a feature transaction F_T by taking the union of the features of all items in T. In other words, $F_T = \{features(I) | I \in T\}$. Once the feature transaction database is obtained, we can apply classical frequent itemset mining algorithms such as APriori[3] with appropriately chosen threshold. However, as pointed out in the example from Section 1, this also suffers from a subtle pitfall of mistaking frequency for importance.

4.2 Adapting Probabilistic Itemset Mining Algorithms.

In this subsection, we describe an intuitive approach that transform the *certain* item transaction database into an *uncertain* feature transaction database.

Attribute Uncertainty Model. The most basic model for describing an uncertain transaction database is the *attribute uncertainty model* [17]. Under this model, each attribute of the tuple is associated with a probability. If we consider a transaction as a tuple where the attributes are $\{I_1, I_2, \ldots, I_k\}$, then each item $I \in \mathcal{I}$ exists in T with an existential probability $P(I \in T)$. This model assumes that the presence of an item is independent of other items in the transaction.

Tag-Cloud Approach. This approach works by transforming the *certain* item transaction database into an *uncertain* feature transaction database. This is an adaptation of the technique used in [7]. We call this as a *tag cloud* [21] based approach (as tag clouds in social media are often generated this way). Each item transaction T is converted to an *uncertain* feature transaction F_T where the existential probability of a feature is computed as the ratio of number of items in which it is present to the total number of all items in the transaction. Formally, $F_T = \{f : \frac{\sum_{I \in T} |\{f\} \cap features(I)|}{|T|} | f \in features(T)\}$. Once the feature transaction database is obtained, we can apply any probabilistic frequent itemset mining algorithm such as UAPriori[9] with appropriately chosen threshold.

While more sophisticated that the previous approach, this algorithm still suffers from the flaws described in the introduction. It treats the most frequent feature(set) as also the most important one. As our counter example pointed out this may not always be valid.

5. COMPUTING FEATURESET LIKELIHOOD

Recall from Section 2 that each item in a transaction was chosen by a user due to a subset of its features. The collection of all such subsets for the entire transaction is its generating featureset. If we have access to this information, frequent featuresets can be obtained by running traditional frequent mining over it. However, since this information is not available, we have to generate the possible featuresets and then evaluate their likelihood to be a generating featureset. We tackle this problem in two stages. We first generate all possible candidate featuresets that could have generated the transaction and then using this information estimate the likelihood for each transaction.

5.1 Generating Candidate Featuresets

In this subsection, our objective is to enumerate all possible featuresets that could have generated the transaction. Since each transaction could have an exponential number of generating featuresets, we use the idea of minimal hitting sets to substantially reduce this number.

Hitting Sets and Generating Featuresets. A key idea in reducing the combinatorial explosion comes from observing the close relationship between generating featuresets and hitting sets. Notice that for a featureset to be a potential generating featureset, it has to necessarily have at least one feature from each item in the transaction.

Given a finite set S (also called as universal set) and a collection C of subsets of S, the *hitting set* for C is a subset $S' \subseteq S$ such that it contains at least one element from each subset in C [12]. A hitting set is *minimal* if none of its proper subsets are also hitting sets. If we treat each item as a set of features and the union of all item features of the transaction as the universal set, then it is easy to see that any generating featuresets is also a hitting set. A further reduction in search space of featuresets can be achieved by observing that typically, users choose an item due to a small number of its features. This parsimonious behavior has been exploited in multiple prior work to generate concise models. This observation motivates us to identify *minimal generating featuresets* by using their relation to minimal hitting sets[16, 14].

Running Example. We can express the relation between transactions and the featuresets as a bipartite graph. The transactions form one partition while featuresets form another. An edge exists between a featureset F and a transaction T when F is a generating featureset for T. Using our running example, this bipartite graph would have 3 transactions and 11 featuresets. However, if we use the minimal hitting set requirement, the graph has just 4 featuresets. Figure 2 shows the resulting graph.

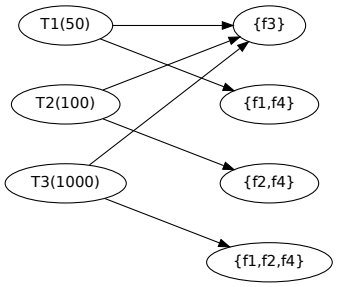


Figure 2: Bipartite Graph with Transaction and Minimal Generating Featuresets

Enumerating Minimal Generating Featuresets.
We describe a simple randomized algorithm to identify minimal hitting sets containing frequent features with high probability. The algorithm starts with an empty candidate hitting set. It then randomly picks a feature and adds it to the candidate. This process is repeated till all items in the transaction are covered resulting in a minimal generating featureset. Each feature is picked with probability proportional to its frequency in uncovered items. In other words, a feature that is present in multiple items has a higher likelihood of being chosen. This enumeration algorithm could be terminated after each featureset in the collection is returned by at least two different invocation of the randomized algorithm This heuristic is also referred to as Good-Turing test. Consider a simple invocation over transaction $T_3 = I_1\{f_1, f_3\}, I_2\{f_3, f_4\}, I_3\{f_2, f_3\}$. The algorithm starts with an empty set. Suppose it picked feature f_3 - it will immediately terminate as all items are covered. Suppose it picked feature f_1. Then items I_2, I_3 are not covered. Since f_3 occurs in both items it is picked with probability $\frac{2}{4} = \frac{1}{2}$ while f_2 and f_4 are picked with probability $\frac{1}{4}$ in the next iteration.

Complexity of Enumerating Minimal Hitting Sets. Identifying a single minimal hitting set requires a time complexity that is linear in the size of all features. However, *counting* all minimal hitting sets is #P-Complete [11]. It is possible to derive tighter bounds with additional information such as the maximum number of features in any item and the maximum transaction size in the dataset by using the idea of parameterized complexity [11].

5.2 Estimating Featureset Likelihood

Given an item transaction T, Section 5.1 identifies the set of featuresets $\mathcal{G}_T$, that could have generated it. However, given the limited information, we do not know the actual generating featureset. We used the featureset uncertainty model to express the probability

that a given featureset generated the transaction T. In this section, we provide an optimization formulation to compute these values.

User-Transaction Interaction Information. Users generate transactions while interacting with items. Such interactions could be represented differently at aggregate or individual levels. At the aggregate level, we do not have access to the id of user who created the transaction. Instead, we treat all transactions to be performed by an "average" user. Such interaction can be represented via an *aggregate interaction vector* v where each component corresponds to the number of times a particular transaction was made. If the user who made the transaction is known, then we could compute, for each user u, an *individual interaction vector* v_u where each component provides the number of times u performed a given transaction. The interaction vector is normalized and has non-negative numbers that add upto 1. For the rest of the section, we assume the average user case while in §6, we describe how to customize the optimization formulation in the presence of user identity.

Transaction-Featureset Transition Matrix. The output of Section 5.1 can be succinctly summarized as a bipartite graph where transactions form one partition while featuresets form another. An edge exists between a featureset F and a transaction T, if F could have generated T. Figure 2 shows the graph for our running example.

We assume a column stochastic matrix $\mathcal{T}$ in which rows correspond to transactions while columns are the featuresets. Each cell $\mathcal{T}_{ij}$ provides the probability that an average user would generate the transaction T_i if she is interested in featureset F_j. We can construct a boolean matrix $\overline{\mathcal{T}}$ that represents the transaction-featureset bipartite graph where $\overline{\mathcal{T}}_{ij} = 1$ if transaction T_i could have been generated by featureset F_j. We also assume that if a featureset F could not have generated a transaction T, then the probability that a user would have generated T to be 0. In other words, $\mathcal{T} \leq \overline{\mathcal{T}}$. There are multiple ways to construct $\mathcal{T}$ from $\overline{\mathcal{T}}$. For example, we can assume a uniform distribution where, given a featureset F, all the transaction that could have been generated by F are chosen uniformly at random. Of course, it is possible to have a non-uniform distribution if we have additional domain knowledge (for example - given an actor, users are more likely to see a movie where the actor starred than one where he doesnt). Our likelihood estimation method is oblivious to the distribution of $\mathcal{T}$.

Featureset Likelihood Vector. As we described in Section 4, it is not possible to accurately identify the likelihood of featureset for a transaction in isolation. It is necessary to utilize the featuresets of other transactions for this purpose. As an example, given a single movie, it is not possible to confidently ascertain which feature caused an user to watch it. However, if we see other movies by the same actor (and with high transaction count), our confidence increases. Due to this reason, we identify the featureset likelihood globally (across all transactions) instead of a single transaction. Intuitively, the global featureset likelihood vector w can be thought of as the "featureset preference vector" of a typical user. In other words, this provides a probability distribution over various featuresets. Frequent featuresets will have a higher value than less frequent featuresets. This vector is stochastic (all entries are non-zero and sum upto 1).

Given the prior notations, we can now formally respecify our generative model for item transactions from Section 2. A typical user has a "featureset-preference" vector that describes a distribution over the featuresets. The user chooses a featureset using that distribution. Once the featureset F_j is choosen, the user looks at the corresponding column in $\mathcal{T}$ and chooses a transaction with probability proportional to $\mathcal{T}_{ij}$. Intuitively, the relationship between various entities can be summarized by:

$$\mathcal{T}w = v \tag{2}$$

Computing Global Featureset Likelihood. From Section 5.1, we obtained the transaction-featureset bipartite graph. Using this information, we can compute the transaction-feaureset transition matrix $\mathcal{T}$ by assuming uniform distribution. We are also provided with the aggregate interaction vector v. Our aim now is to identify the featureset likelihood vector w.

Notice that typically, the number of featuresets far outnumber the number of trasactions. Hence the linear system of equations expressed by Equation 2 is overdetermined and has no solution. We can define an error metric based on the reconstruction error, $Error(v - \mathcal{T}w)$. For the purpose of our paper, we use the L_2 error metric. Our problem boils to finding the best w vector that minimizes $||v - \mathcal{T}w||_2$.

Algorithm 1 FFM-AVG

1: **Input:** $\mathcal{IT}$
2: Compute aggregate interaction vector v from $\mathcal{IT}$
3: $\forall T \in \mathcal{IT}$, generate candidate featuresets
4: Form transction-featureset bipartite graph $\overline{\mathcal{T}}$ and estimate $\mathcal{T}$
5: $constraints = \{\ \forall i\, w_i \geq 0, ||w||_1 = 1\ \}$
6: $w = \underset{w}{\text{argmin}}\ ||v - \mathcal{T}w||_2$ subject to $constraints$
7: **return** w

The solution vector w must minimize the reconstruction error and must also satisfy some constraints. Algorithm 1 provides a pseudocode for the problem. We model this problem as a constrained optimization problem with non negativity and stochastic constraints. Specifically, the optimization with L_2 metric corresponds to a constrained least squares problem with stochasticity constraints. Due to how the objective function is defined, this falls under a subset of convex optimization problem for which optimal solutions can be computed efficiently.

Complexity. The constrained optimization problem has a worst case complexity of $O(N^3)$. However, there exist very efficient iterative algorithms that can obtain the optimal solution in few iterations [8].

6. MINING FREQUENT FEATURESETS

Let us recap what we have achieved so far. We started with a database of item transactions with the objective of identifying frequent featuresets. Using a novel uncertainty model, we expressed each item transaction as a collection of featuresets that could have generated it. Using this transaction-featureset bipartite graph as a base, we used a constrained least squares approach to identify the global likelihood for each featureset. In this section, we use this information to identify the frequent featuresets. We first describe an algorithm that performs mining directly over the featureset uncertainty model. We then design an efficient yet approximate algorithm that transforms the featuresets and their likelihoods into feature transactions which can then passed to any state of the art probabilistic itemset mining algorithms.

6.1 Exact Algorithm for Mining Frequent Featuresets

In this subsection, we describe an exact algorithm *FFM-EXACT* for identifying all frequent featuresets. This algorithm takes as input a single featureset F and computes whether F is a probabilistic

frequent featureset. Our algorithm is based on dynamic programming and runs in polynomial time and is adapted from the approach first described in [22].

Frequent Featuresets and Probabilistic Heavy Hitters. There exists a close parallel between the concept of frequent featureset and that of probabilistic heavy hitters in uncertain data [22]. Given an uncertain database, an item t is considered as a (ϕ, τ)-probabilistic heavy hitter if the probability that t is heavy hitter (i.e. occurs more than fraction of ϕ in a possible world) is greater than τ. We can immediately see that if we treat each featureset F as an item (and compress other featuresets of a transaction to $\overline{F}$), then verifying whether F is a (*sup(X), pft*)-heavy hitter corresponds to finding if the featureset is probabilistically frequent. Our exact algorithm *FFM-EXACT* takes this approach.

This algorithm takes as input a single featureset F and the per-transaction featureset likelihood estimated from Section 5. It then converts each item transaction T into an uncertain transaction with two possible items F and $\overline{F}$. The existential probability (for a featureset F and for a transaction T) $Pr_T(F)$ is set to estimated featureset likelihood if F was a generating featureset of T. Else the value was set to 0. The existential probability of $\overline{F}$ is computed as $1 - Pr_T(F)$.

Given this information, we could use the algorithm in [22] to estimate the probability that F is a frequent itemset. This algorithm is based on dynamic programming. We create a two dimensional table B_F with N rows and N columns. The cell $B_F(i, j)$ is interpreted as the probability that F was the generating featureset in exactly i item transactions out of the first j item transactions. Using this interpretation, the table can be filled as follows.

Base Case:

$$B_F[0,0] = 1$$
$$B_F[i,0] = 0 \quad (i \geq 1)$$
$$B_F[0,j] = \begin{cases} B_F[0,j-1] & \text{if } F \in T, j \geq 1 \\ B_F[0,j-1](1 - Pr_{T_j}(F)) & \text{if } F \notin T, j \geq 1 \end{cases} \tag{3}$$

Induction Step:

$$B_F[i,j] = \begin{cases} B_F[i,j-1] & \text{if } F \notin T \\ B_F[i,j-1](1 - Pr_{T_j}(F)) + & \\ \quad B_F[i-1,j-1]Pr_{T_j}(F) & \text{if } F \in T \end{cases} \tag{4}$$

Once the table is filled the probability that F is a frequent featureset can be computed as in [22]. This step is repeated for each featureset and runs in $O(N^2)$ per featureset.

6.2 Approximate Algorithm for Mining Frequent Featuresets

While the previous algorithm is exact and produces accurate results, it is prohibitively expensive. In this subsection, we design an *approximate* algorithm *FFM-APPROX* by sacrificing some accuracy in favor of dramatically improved efficiency.

From Generating Featuresets to Feature Transactions. Our next step is to use this collection of transactions and the featuresets with likelihood weights to identify the frequent featuresets. Unfortunately, there exist no algorithm to identify the frequent featuresets. Most uncertain frequent mining problems work over attribute uncertainty model necessitating a transformation from featureset uncertainty model to attribute uncertainty model. This transformation might seem counterintuitive - we are moving from featureset uncertainty model (which is quite expressive), to a simpler uncertainty model. Further, it might seem that we are abandoning all the results of our expensive pre-processing. However, as out later experiments show, performing frequent featureset mining over this relaxed dataset provides a higher quality results than transforming to attribute uncertainty model directly.

In this section, we will focus on taking item transactions along with their candidate generating featuresets into a simpler feature transaction where each feature has an existential probability associated with it. Higher the probability, the more likely its corresponding feature to generate more items in the item transaction.

It is possible to generate feature transactions at multiple levels of granularity.

Average User - Per Transaction. This is the simplest transformation and the most generic. This is applicable in the case where the original item transaction database did not have any mechanism to identify the user who made the transaction. Further, in this approach we convert each item transaction into a corresponding feature transaction.

Algorithm 2 FFM-APPROX

1: **Input:** $\mathcal{G}$:< $\mathcal{T}$,$\mathcal{F}$,$\mathcal{E}$,$\mathcal{P}$ > - bipartite graph where $\mathcal{P}$ represents the likelihood distribution on $\mathcal{F}$
2: $\mathcal{FT} = \{\}$
3: **for all** $T \in \mathcal{IT}$ **do**
4: Collect generating featuresets GF_T=$\{F_i \in \mathcal{F}$ — $(F,T) \in \mathcal{E}\}$
5: Normalize the weights of the generating featuresets, $\forall F_i \in GF_T, P(F_i) = P(F_i)/\sum_{F_j \in GF_T} P(F_j)$
6: Initialize a feature transaction, $TmpF_T = \cup F_i | F_i \in GF_T$
7: Update weights of individual features, $\forall f_i \in TmpF_T$, $f_i.weight = \sum_{F_j \in GF_T} P(F_j) * F_j(f_i)$ if $f_i \in F_j$, 0 otherwise
8: $\mathcal{FT} = \mathcal{FT} \cup \{TmpF_T\}$
9: **end for**
10: Invoke uncertain frequent mining on $\mathcal{F}$

Algorithm 2 works as follows. First, for each item transaction T, we isolate all its generating featuresets from the transaction-featureset bipartite graph. We then normalize the featureset likelihoods so that they sum up to 1. The updated values provide, for each featureset, the probability that it could have generated T given other transactions. We then convert this to a feature transaction as follows. For each feature f in T, we identify all the generating featuresets of T in which it is part of. The existential probability of the feature f is given by the summation of the normalized likelihoods of the relevant featuresets.

Running Example. Recall that Figure 2 gave the minimal generating transactions for the dataset. After running the constrained optimization, the global likelihood for each featureset is computed. Figure 3 shows an arbitrary subgraph for illustration purposes.

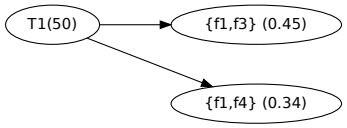


Figure 3: Bipartite Graph with Featureset Probabilities

Let us convert item transaction T_1 to a feature transaction. The featuresets associated with T_1 are $\{f_1, f3\}$ and $\{f1, f4\}$ with values 0.45 and 0.34 respectively. We first normalize the values so that updated values are 0.569 and 0.431. In other words, the first featureset had a likelihood of 0.569 to have generated T_1. We now

create the feature transaction from these normalized values. In this case, $\{f_1\}$ was present in both featuresets, its existential probability is $0.569 + 0.431 = 1$. For the other features, their existential probabilities are computed based on the single featureset they each belonged to. The final feature transaction is $\{f_1 : 1.0, f_3 : 0.569, f_4 : 0.431\}$.

Per User - Per Transaction. If we have information that could identify the user who made the transactions, we could design a user-specific feature transaction. In this approach, we start with the subset of item transactions performed by the user. Then we generate the candidate featuresets, identify their likelihood and use the previous method to convert to feature transaction. The major advantage is that if there is an item transactions made by two different users, the previous method would generate identical feature transaction. However, in this case, they could potentially generate different feature transactions.

Per User. Traditional frequent featuresets are executed over the entire dataset. However, it is also possible to identify frequent featuresets across users. A major advantage is that this approach provides featuresets popular among a large fraction of the user population as opposed to featurests popular among a large fraction of transactions. This approach alleviates the effect of users who make a lot of identical transactions hence generating irrelevant frequent featuresets. We start from the subset of item transactions performed by the user. We then identify the candidate featuresets for all transactions and identify their global likelihood. Notice that the likelihood Then, we create a *single* feature transaction for the user. For each feature that is present in some transaction made by the user, its existential probability is the sum of featuresets in which the featureset is present.

Frequent Featureset Mining. Once we have converted the item transactions into feature transactions, they could be passed to any state-of-the-art uncertain frequent itemset mining algorithms [9, 15, 1] to obtain frequent featuresets. As the experiment section will confirm, the featuresets obtained are of higher quality than direct adaptations of certain or uncertain frequent itemset algorithms.

7. SCALING FEATURESET MINING

The previous sections described the various stages involved in taking a dataset of item transactions and processing it to identify the frequent featuresets. These algorithms are feasible for datasets with a relatively small number of features or items. Also recall from §5 that the worst case complexity of finding the likelihood is cubic in the size of transactions. In this section, we describe how to overcome the two major scalability challenges - large number of candidate generating featuresets and large size of item transaction databases. Surprisingly, the solution to both the problem rely on a common trick - sampling.

7.1 Sampling Transaction-Featureset Bipartite Graph.

Recall that one of the major challenges in identifying frequent featuresets is to enumerate all the major candidate featuresets of each transaction. This was similar to enumerating all the minimal hitting sets. In the worst case, this number could be exponential. This issue is amplified by the fact that the same process has to repeat for each item transaction in the database. Hence, in order to scale the algorithm, we have to identify the most important featuresets and retain only them. Any featureset that does not have a high likelihood could be removed.

We use a combination of heuristics to scale the algorithm. First, we relax the constraint that the generating featureset has to be minimal. This might seem counterintuitive as the number of non minimal generating featuresets far outnumbers that of minimal ones. However, this relaxation allows us to build featuresets that are simultaneousaly related to multiple transactions. We adapt the existing randomized algorithms described in §5.1 for this purpose. Given the expected support threshold, we identify the minimum number of transactions that any candidate generating featureset has to cover. For example, it might be the case that for a featureset to be frequent, it has to cover at least 100 transactions. The heuristic starts with a hitting set containing all the features in $\mathcal{F}$. This is obviously a valid featureset touching all transactions. We randomly drop features from it till the number of transactions it covers reach the threshold (say 100). We repeat the process as long as necessary to cover all the transactions in the database. We can observe that each iteration of this approach results in a featureset that covers at least 100 transactions. These featuresets also contains features that are, intuitively, more likely to be in the final frequent featuresets. A bottom-up variant of this heuristic is also possible where we start with an empty set and add features till it covers at least 100 transactions. Refer to Algorithm 3 for pseudocode. This approach, in the worst case, degenerates to the algorithms we described in previous sections.

Complexity of FFM-APPROX-SAM1. From the pseudocode in Algorithm 3, it is possible to analyze its worst case complexity. From §5, recall that the complexity of FFM-APPROX is $O(N^3)$ where N is the number of transactions. Other operations in the algorithm can be ignored for asymptotic analysis. This implies that the algorithm FFM-APPROX-SAM1 has a time complexity of $O(N^3)$. In practice, the scalable variant is extremely efficient.

Algorithm 3 FFM-APPROX-SAM1

Input: $\mathcal{G}$: bipartite graph, r: max featuresets, t: threshold
Candidate featuresets $\mathcal{H}$ = {}
for index=1 **to** r **do**
  Candidate hitting set $hs = \{\}$
  randomly add features from $\mathcal{F}$ until hs hits at least t item transactions
  $\mathcal{H} = \mathcal{H} \cup hs$
end for
$\mathcal{G}$ = Build-Bipartite-Graph($\mathcal{IT}, \mathcal{H}$)
w = FFM-APPROX($\mathcal{IT}$)
$\mathcal{FT}$ = FEATURE-TRAN-GEN($\mathcal{G}$)
FFS = frequent featuresets from ($\mathcal{FT}$)
return FFS

7.2 Sampling Item Transactions

An orthogonal approach to scale is based on sampling the item transaction database so as to get a smaller sample on which the algorithms are feasible. Intuitively, we pick a random sample of the database and run the algorithms described in previous sections over it. If the sample obtained is representative, then the frequent featuresets obtained with a scaled down threshold would also be frequent in the entire database. A key issue with this sampling strategy is the possibility of errors. Specifically, it is possible to get both *false negative* (when a featureset is frequent in the database but not in the sample) and *false positive* when a spurious frequentset is frequent in the sample but not in original dataset.

There are multiple strategies to reduce the number of false positives and negatives ranging from using a lower threshold, using multiple chunked samples instead of a single sample etc. One of

the elegant algorithms to remove the false positives or negatives completely is the one proposed by Toivonen et al. [19]. We adapt a variant of this algorithm for our paper. We start with obtaining a random sample and run our algorithms over it. Once the frequent featuresets are identified, we collect the featuresets in the *negative border*. These are featuresets that are not frequent by themselves but *all* their immediate subsets are. We then make a pass over the entire dataset and verify the expected support of all the frequent featuresets identified from the sample and also their negative border. The set of frequent featuresets identified in the second pass are exactly the solution to our original problem. Refer to Algorithm 4 for pseudocode.

Algorithm 4 FFM-APPROX-SAM2

1: **Input:** $\mathcal{IT}$
2: Generate sample $\mathcal{S}$ from $\mathcal{IT}$
3: $\mathcal{G}_\mathcal{S}$ = transaction-featureset bipartite graph for $\mathcal{S}$
4: w = FFM-APPROX($\mathcal{IT}$)
5: $\mathcal{FT}_\mathcal{S}$ = FEATURE-TRAN-GEN($\mathcal{G}_\mathcal{S}$)
6: FFS_S = frequent featuresets from ($\mathcal{FT}_\mathcal{S}$)
7: FFS_S^- = Compute negative border for FFS_S
8: FFS = Filter non frequent featuresets from $FFS_S \cup FFS_S^-$
9: **return** FFS

Complexity of FFM-APPROX-SAM2. The analysis of FFM-APPROX-SAM2 follows from that of FFM-APPROX-SAM1. The dominant factor is the procedure FFM-APPROX which takes time $O(N_S^3)$. Unless the sample size is very small, this provides the time complexity. Of course, the scalable variant is in practice extremely efficient.

8. EXPERIMENTS

In this section, we provide an extensive experimental evaluation of the efficiency and effectiveness of our proposals in mining hidden frequent featuresets on real and synthetic datasets. The experimental results confirm the superiority of our approach over direct adaptations of existing frequent itemset mining algorithms.

System Configuration. All our algorithms are implemented in C++. Experiments were conducted on Linux Ubuntu 13.04 machine, Intel Core i5 processor, 64 bit machine with 8 GB RAM. Timing values are taken by averaging over twenty runs.

8.1 Datasets

We used three different datasets to evaluate our algorithms. Two of them are real-world dataset while the third is a synthetic data allowing us to evaluate the scalability of our algorithms.

AJE dataset : This dataset consists of 2103 news articles published between April 2012 and February 2013 on Aljazeera english (AJE) website[1] - one of the most influential news media in the MENA region[2]. Each article comes with a set of comments (389k in total) posted by 35k different users from 179 different countries. We characterized each article by its features (i.e. topics, persons and locations) extracted using Open Calais[3]. We also evaluated other entity extraction libraries such as Alchemy and Stanford NER but found the results produced by all three services were remarkably similar. On average, each article has 7.5 features distributed as follows: 1.42 topics, 2.58 person names and 3.5 locations names (countries and/or cities).

An item transaction is defined as the set of articles commented by a user uid on the same date d, augmented with the comments a user posted on each article i.e. $T =< tid, uid, (a_1, c_1), \ldots (a_k, c_k), d >$, where c_i could be the concatenation of all comments posted by uid on a_i and data d. Grouping the data by users then by dates results in 15358 transactions. See Table 2 for further details.

AJE Ground Truth. We propose to use user comments as a proxy to uncover the hidden features that interested a user while reading an article. Given an article a with a set of features $features(a)$, a comment c posted by user u on article a, we assume that the features that most interested the user u are those of the intersection $features(a) \cap features(c)$. The ground truth for a transaction is the set of ground truth for all articles. While this ground truth is an approximation of the real ground truth (which ideally would be obtained by surveying users), we found that our approach is an efficient and automatic way to extract the ground truth. We conducted an user study where we evaluated different mechanisms to identify ground truths - our approach identified more entities than other alternate methods. Finally, the set of ground truth frequent featuresets (GT) is obtained by running a deterministic frequent itemset mining algorithm on the obtained feature transactions.

Synthetic dataset SD : The synthetic dataset was created to test the various facets of our algorithm. It was generated in two phases. In the first, we used IBM Quest Generator to generate 1000 structured items over 100 features with average feature size of 4. (here dataset transaction corresponded to items while dataset items corresponded to features). Once the structured items are known, we then create another dataset for the actual evaluation purposes. The output was a collection of feature transactions that also forms the ground truth. For each feature in the transaction, we chose the corresponding item (of item transaction) uniformly at random. For eg, suppose the feature transaction was $\{f_1, f_2\}$. We now look at all possible items with f_1 (resp f_2) and choose an item at random. items with their presence.

Last.fm Dataset LF: Last.fm [4] is a music website where users could listen to songs from internet radio stations or from their portable music devices. Songs corresponds to items. Each song is described using multitude of features including artist, genre, albums, record labels and semi-structured information via tags. The set of songs played by a user in a session correspond to the transaction. Last.fm provides a scrobbling API that allows programs to send and receive information about tracks listened. We built a Chrome extension that used Last.fm's API to monitor the set of tracks listened by the user. We recruited (via Reddit[5]) more than 2000 volunteers to install our extension. Our extension provides a gamified interface where it periodically inquires users about which of the track's features lead them to listen to it. The users were also allowed to enter free-form content using tags or other detailed description. These formed the ground truth over which our algorithms were evaluated.

Table 2: Characteristics of Dataset

Dataset	#Trans.	#Items	#Features	Avg Len
AJE	$15K$	2103	459	1.5667
SD(T10I4D100K)	$100K$	1000	100	10
LF	$40K$	5644	919	18

[1] http://www.aljazeera.com
[2] MENA: Middle East and North Africa
[3] http://www.opencalais.com/
[4] http://www.last.fm/
[5] http://www.reddit.com/

8.2 Evaluation metrics

The evaluation of our algorithms is quite tricky due to the limited amount of information available. Specifically, it is not makes sense to use traditional metrics used for exact itemset mining. Instead, we used measures that are commonly used in approximate frequent itemset mining [13]. A brief description is given below.

Recoverability. Recoverability measures how well a pattern mining algorithm recovers the ground truth featuresets (GT). For each ground truth featureset pattern GT_i, we first identify the frequent featureset FFS_i generated by our algorithm that best matches with it (based on the number of common features cf_i they share). In other words, the recoverability of GT_i is the largest percent of an featureset found by any pattern FFS_i that is associated with GT_i. This is performed as a weighted average as matching with a larger pattern counts much than matching with smaller patterns.

$$Recoverability = \frac{\sum_{i=1}^{|GT|} cf_i}{\sum_{i=1}^{|GT|} |GT_i|} \quad (5)$$

Spuriousness. It is possible for an algorithm to get a high recoverability by returning large featuresets. Spuriousness is a metric complementary to recoverability - it measures the number of features in the pattern that are not associated with original matching pattern. Further, precision can be computed as 1.0-spuriousness. For each FFS say FFS_i, we first identify the ground truth featureset GT_i, which share maximum number of common features (denoted as cf_i) between them. The number of spurious items for each FFS is, $|FFS_i|$ - $|cf_i|$. The equation to calculate the *spuriousness* over whole FFS is given below.

$$Spuriousness = \frac{\sum_{i=1}^{|FF|}(|FF_i| - cf_i)}{\sum_{i=1}^{|FF|} |FF_i|} \quad (6)$$

Significance. This metric combines both *recoverability* and *Spuriousness* in the same way F-measure does with *precision* and *recall*.

$$Significance = \frac{2 * (Recoverability * (1 - Spuriousness))}{(Recoverability + (1 - Spuriousness)} \quad (7)$$

Redundancy. This metric mitigates the effect of producing relevant but redundant FFS (i.e. frequent sets that are subsets of other frequent sets). Redundancy is measured by creating a matrix, M of size $|FFS|X|FFS|$. Each entry in the matrix,ffs_{ij} denotes the number of common items between FFS_i and FFS_j. Then we compute the sum of the upper triangular matrix of M and subtracting the diagonal entries to estimate the redundancy. This measure doesn't take the average over the number of FFS. Hence, it implicitly penalize if the number of FFS is too large.

$$Redundancy = \frac{\sum_{i,j=1\ldots|FFS|,j\succ i} ffs_{ij} - \sum_{i=1\ldots|FFS|} ffs_{ii}}{2} \quad (8)$$

8.3 Qualitative Evaluation

For the qualitative purposes, we test two algorithm. BASELINE is a simple tag cloud based approach defined in Section 4. We did not test the deterministic baseline as it consistently underperformed BASELINE. The frequent featureset mining algorithm *FFM-APPROX* treats all transactions are made by a single user. We evaluate the exact, approximate and sampling based variants of our algorithm. After identifying the frequent featuresets using baseline and our algorithms, we evaluate their quality using the evaluation metrics described previously.

Experimental Observations: Figures 4(a)-4(l) show how our algorithms perform against baseline for the three datasets. The major observations are as follows: (a) Our algorithms consistently out perform BASELINE for larger value of support (b) The exact algorithm `FFM-EXACT` have a higher score than the approximate versions. (c) The sampling based algorithms perform slightly worse than the approximate variants. `FFM-APPROX-SAM1` which is designed for speed has a lower accuracy than `FFM-APPROX-SAM2` that is optimized for accuracy.

We vary the minimum support from 0.1 to 0.5 and measure its impact over the evaluation metrics. Figures 4(a), 4(b) and 4(c) show the corresponding impact over recoverability. Higher values of recoverability is desirable. We can see that the recoverability of the algorithms increase as minimum support decreases. This expected behavior is due to the high number of FFS that are discovered at lower values of support, increasing the number of recovered GT. Our algorithms out performs BASELINE for higher values of support. Figures 4(d), 4(e) and 4(f) show the spuriousness scores achieved at different support values. A lower value of spuriousness is desired. The figures show that our algorithms obtain a significantly lower value of spuriousness than BASELINE. Higher values are desirable for significance while lower values are desirable for redundancy. Figures 4(g)-4(l) show that our algorithms have a superior performance for both significance and redundancy.

8.4 Quantitative Evaluation and Scalability

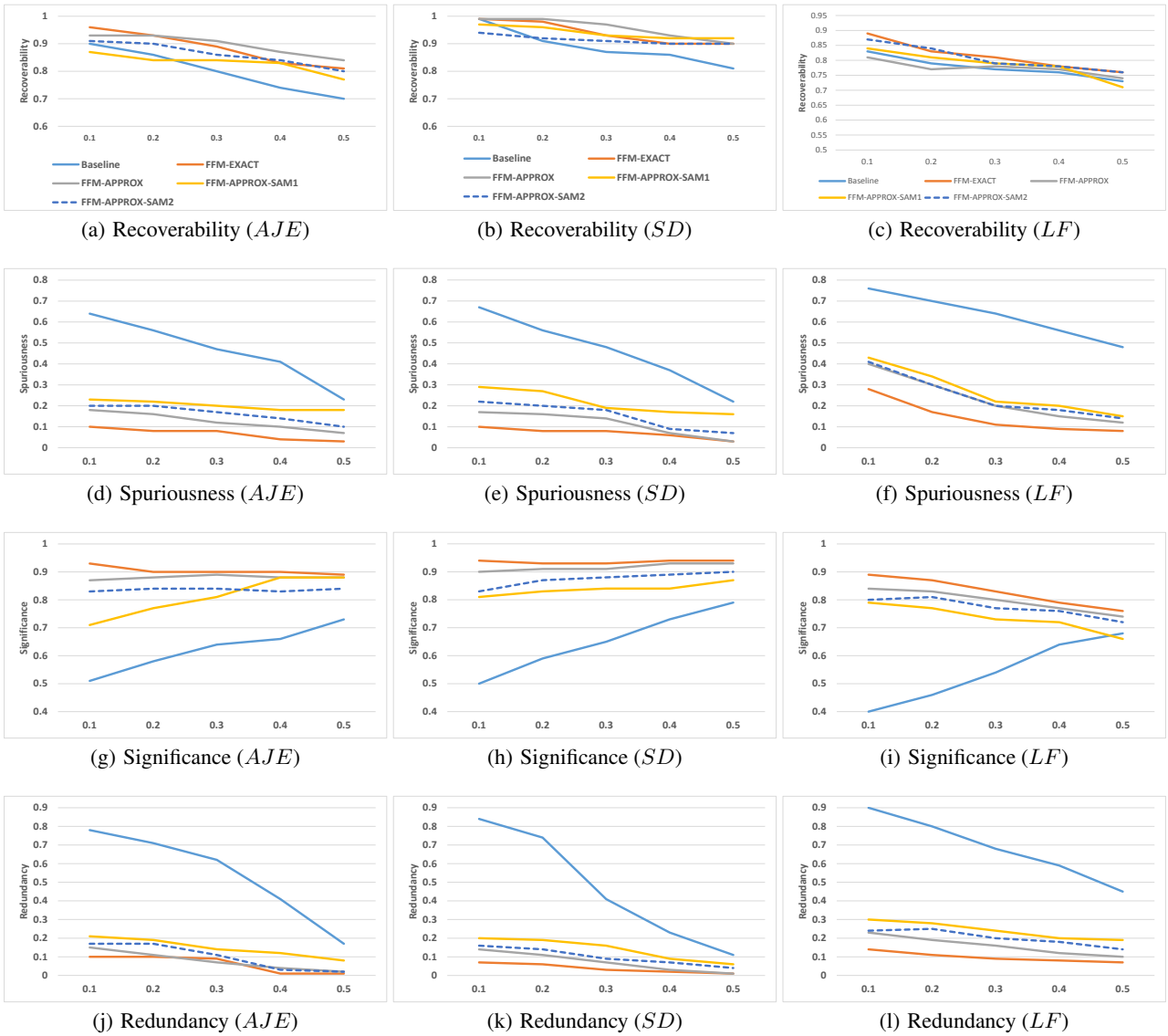
Scalability. Figures 5(a)-5(c) show the runtime performance of all our algorithms. We measure three parameters - number of transactions (N), number of items (n) and number of features (m). As expected, the exact algorithm takes prohibitive amount of time and for large datasets, it took more than two days. The approximate variant is much faster than the exact version while the two sampling variants are orders of magnitude faster. We can also see that the number of transactions and features have a higher impact on running time than the number of items. This is to be expected as the major factor in the runtime is the number of featuresets which are directly impacted by the number of features.

Robustness of Sampling: In this set of experiments we measure the robustness of our sampling algorithms. We utilize the theoretical results from [19] as a heuristic to determine the minimum sample size. We seek to obtain a *representative* sample[19] by ensuring that, given an itemset S, the probability that the difference between S's relative frequency in the database and the sample varies by atmost a constant ϵ is less than a constant δ. Given ϵ, δ, [19] provides the minimum sample size that must be obtained regardless of database size. Figures 6(a)-6(c) show the robustness of our algorithms. Once the sample size exceeds 30K (minimum sufficient size for $\epsilon = \delta = 0.01$), the quality does not dramatically improve with higher sample size. Figures6(a), 6(b) show the results for recoverability and Significance respectively. The results of other metrics were similar and not included to conserve space. Figure 6(c) shows that given a fixed sample size as suggested by [19] is sufficient to reach a good accuracy regardless of the size of the database.

9. RELATED WORK

While there has been extensive work on Frequent itemset mining starting with [4], most of the proposed approaches and techniques assume the atomicity of items and hence aim at identifying frequent itemsets by mining transactions over items. To the best of our knowledge, we are the first to solve the problem of identifying hidden frequent featuresets by mining observed transactions using a novel featureset uncertainty model.

Uncertain Frequent Itemset Mining. Chui and al. [9] were the first to investigate mining frequent itemsets over uncertain transac-



(a) Recoverability (AJE) (b) Recoverability (SD) (c) Recoverability (LF)

(d) Spuriousness (AJE) (e) Spuriousness (SD) (f) Spuriousness (LF)

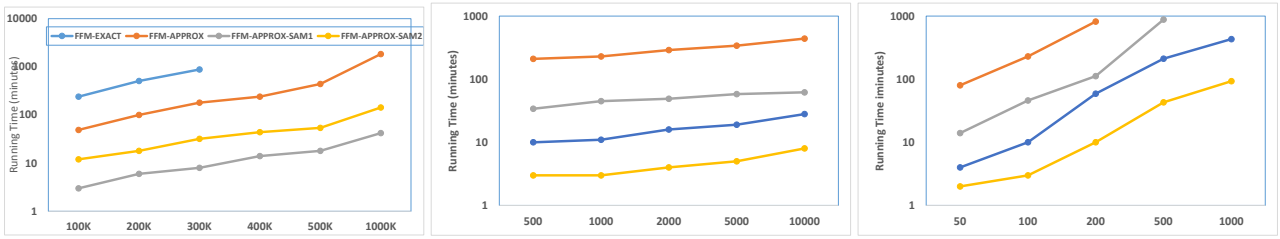
(g) Significance (AJE) (h) Significance (SD) (i) Significance (LF)

(j) Redundancy (AJE) (k) Redundancy (SD) (l) Redundancy (LF)

Figure 4: Qualitative Evaluation for AJE, SD and LF Datasets with Varying Minimum Support

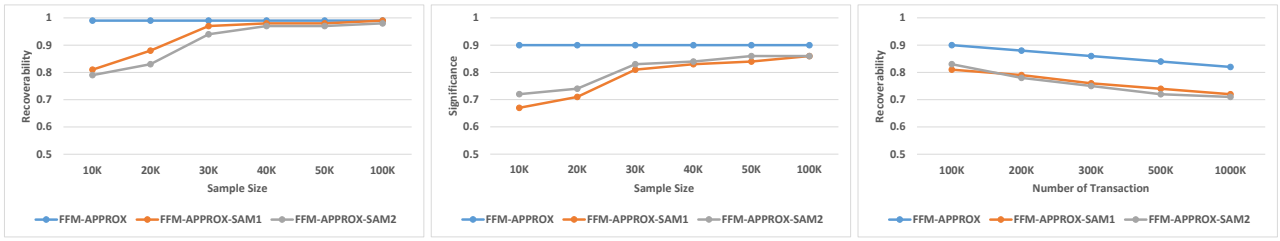
tions. Since then, an important amount of work has been done in this area (see [2] for a survey). There are two main approaches for mining uncertain frequent itemsets: expected support and probabilistic models, both of which consider the support as a random variable but with different interpretations. In expected support approaches [9, 15, 1], an itemset is frequent iff its expected support is greater or equal than a predefined threshold. Several well-known frequent itemset mining algorithms have been adapted to deal with the expected support such as UAPriori [9], UFP-Growth [15], and UH-Mine [1]. On the other hand, probabilistic frequent itemset mining approaches rely on the concept of *frequentness probability* which denotes the probability of an itemset support to be greater than a predefined minimum support [7, 18]. Only itemsets with frequentness probability greater than the minimum support are considered as frequent. This is typically solved using dynamic programming [7] or divide and conquer [18]. Tong et al. showed that these two definitions are equivalent [20].

Uncertain Model in Probabilistic Database. There are three different models to represent uncertainty in relational databases: *tuple uncertainty*, *attribute uncertainty*, and *xtuple* uncertainty. In *tuple uncertainty* model, each tuple is associated with a score reflecting the probability of that tuple to exist in the database [10]. In *attribute uncertainty* model, the uncertainty is more fine grained where a probability score is assigned to each attribute value in a tuple [17]. Notice that both models are mapped into probability distribution over all the *Possible World* [5] where each *Possible* world is an deterministic instance of the database. The concept of *xtuples* is used in the ULDB model proposed in [6]. An xtuple could be seen as a probability distribution over a set of mutually exclusive tuples. Uncertain itemset mining approaches follow either the tuple uncertainty model [18] where a probability score is associated with each transaction, or attribute uncertainty model [7] where probabilities are associated to each item in a transaction.



(a) Number of Transactions(N) (b) Number of Items(n) (c) Number of Features(m)

Figure 5: Effect on Running Time



(a) Recoverability Vs Sample Size (b) Significance Vs Sample Size (c) Recoverability Vs No. Of Transactions

Figure 6: Robustness of Sampling Algorithms

10. FINAL REMARKS

In this paper we study the novel problem of mining hidden frequent *featuresets* from item transactions. We motivated this novel problem with several illustrative examples and introduced a featureset uncertainty model. We developed a constrained least squares based approach to solve the problem of learning generating featureset likelihoods. We also developed two heuristics based on sampling to scale up our algorithm to real world problem sizes.

Acknowledgment

The work of Saravanan Thirumuruganathan, Habibur Rahman and Gautam Das is partially supported by NSF grants 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board, and grants from Microsoft Research and Nokia Research.

11. REFERENCES

[1] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *SIGKDD*, pages 29–38. ACM, 2009.

[2] C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE TKDE*, 2009.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, New York, NY, USA, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[5] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992. IEEE, 2008.

[6] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[7] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic frequent itemset mining in uncertain databases. In *SIGKDD*, pages 119–128. ACM, 2009.

[8] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[9] C. K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *PAKDD*, pages 47–58, 2007.

[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4):523–544, 2007.

[11] P. Damaschke. The union of minimal hitting sets: parameterized combinatorial bounds and counting. In *STACS*, pages 332–343, 2007.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[13] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *KDD*, pages 301–309, 2008.

[14] T. Hofmann. Learning what people (don't) want. In *EMCL*, EMCL '01, pages 214–225, 2001.

[15] C. K.-S. Leung, M. A. F. Mateo, and D. A. Brajczuk. A tree-based approach for frequent pattern mining from uncertain data. In *AKDDM*. Springer, 2008.

[16] R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, Apr. 1987.

[17] P. Sen, A. Deshpande, and L. Getoor. Representing tuple and attribute uncertainty in probabilistic databases. In *Data Mining Workshops, ICDM*, pages 507–512. IEEE, 2007.

[18] L. Sun, R. Cheng, D. W. Cheung, and J. Cheng. Mining uncertain data with probabilistic guarantees. In *SIGKDD*, pages 273–282. ACM, 2010.

[19] H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145, 1996.

[20] Y. Tong, L. Chen, Y. Cheng, and P. S. Yu. Mining frequent itemsets over uncertain databases. *VLDB*, 2012.

[21] P. Venetis, G. Koutrika, and H. Garcia-Molina. On the selection of tags for tag clouds. In *WSDM*, 2011.

[22] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. SIGMOD '08, 2008.