The Difference and the Norm

Characterising Similarities and Differences between Databases

Kailash Budhathoki and Jilles Vreeken

Max-Planck Institute for Informatics and Saarland University, Germany
{kbudhath,jilles}@mpi-inf.mpg.de

Abstract. Suppose we are given a set of databases, such as sales records over different branches. How can we characterise the differences and the norm between these datasets? That is, what are the patterns that characterise the general distribution, and what are those that are important to describe the individual datasets? We study how to discover these pattern sets simultaneously and without redundancy – automatically identifying those patterns that aid describing the overall distribution, as well as those pointing out those that are characteristic for specific databases. We define the problem in terms of the Minimum Description Length principle, and propose the DIFFNORM algorithm to approximate the MDL-optimal summary directly from data. Empirical evaluation on synthetic and real-world data shows that DIFFNORM efficiently discovers descriptions that accurately characterise the difference and the norm in easily understandable terms.

1 Introduction

Suppose we are given a set of databases, such as the sales records over different branches of a chain. How can we characterise the differences and the norm between these datasets? That is, what are the patterns that are common to all databases, and what are those that are important to characterise the individual databases? For example, whereas *bread* and *butter* may be an important pattern in all stores, *pasta* and *ketchup* may only be descriptive for the store on campus. When we mine only the complete data we risk missing the locally important patterns, and when we mine the databases individually we risk missing the bigger picture. We want to discover all important patterns, without redundancy, and such that it is clear which databases they are characteristic for.

More in particular, given a set of databases, we want to discover a *set of patterns* per database or combination of databases that the user is interested in. These pattern sets should only include patterns that are descriptive for the databases associated with the set, and overall these sets should be as non-redundant as possible. That is, together these pattern sets should succinctly summarise the given data.

We formalise this goal in terms of the Minimum Description Length principle [5,13]. That is, we define the best model as the *set of pattern sets* that describes the data most succinctly without loss. By this objective, a pattern will only be included in the model if it simplifies the description – if it aids compression. This means our model will not be redundant, nor will it include noise.

To describe an individual database we only have to consider those patterns that are associated with that database. This allows us to associate patterns with the databases

they are most characteristic for. As characteristic does not necessarily mean 'same frequency', we do not want to punish patterns for having different frequencies in the different databases they are associated with. To avoid such undue bias we carefully construct a score for this setup using prequential coding, a form of Refined MDL [5].

To discover good model directly from data we introduce the DIFFNORM algorithm. DIFFNORM iteratively searches for that pattern that maximally simplifies the current description. To this end it searches for those itemsets X and Y in its model that are most frequently co-used to describe the same transaction, and considers their union as a candidate pattern. The intuition is that these codes for X and Y are redundant, and that by introducing that $X \cup Y$ to the model the description will become more succinct.

Empirical evaluation on synthetic and real-world data shows that DIFFNORM efficiently discovers descriptions that in easily understandable terms accurately characterise the difference and the norm. On synthetic data it recovers the ground truth of both local and global patterns, without picking up on noise. On real world data it discovers succinct and interpretable pattern sets that characterise the split over the data well.

The remainder of this paper is organised as usual. For readability we postpone details on selected derivations to Appendix A.

2 Related Work

Comparing two or more transaction databases is a common task, yet there exist surprisingly few techniques that can characterise similarities and differences of databases in easily understandable terms. Traditional frequent pattern [1] as well as supervised pattern mining approaches [10] for example, discover far too many patterns for the result to be interpretable. Pattern set mining circumvents the pattern explosion [17]. Existing unsupervised methods such as TILING [4], SLIM [14], and MTV [8] only characterise one database at a time, while supervised methods only describe what sets databases apart. Running these algorithms on multiple (combinations of) databases and comparing the results does not work in practice – small differences in the data distribution can lead to very different pattern sets which are difficult to compare.

Earlier, Vreeken et al. [16] proposed a dissimilarity measure for transaction data based on KRIMP [17]. The main idea is to infer a pattern set per database, and then measure how many bits more we need to describe the other databases with these patterns – the more similar the data, the small the difference. Here, on the other hand, we are interested in characterising all databases at the same time, without redundancy.

In Joint Subspace Matrix Factorization (JSMF) the goal is to discover the common subspace between the two datasets, as well as those that are representative of the specific datasets. Most relevant, as it considers binary databases, is Joint Subspace Boolean Matrix Factorization (JSBMF) [9]. To avoid overfitting, it requires the user to specify the number of patterns per pattern set. Our approach is parameter free. Moreover, as JSMF is defined for pairs of databases, and not trivially extendable to arbitrary combinations of databases, it cannot simultaneously and without redundancy find the patterns over multiple subspaces.

3 Preliminaries

In this section we discuss preliminaries and introduce notation.

3.1 Notation

We consider transaction data. Let $\mathcal{I}$ be a set of items, e.g. products for sale in a store. A transaction $t \in \mathcal{P}(\mathcal{I})$ then corresponds to the set of items a customer bought. A database D over $\mathcal{I}$ is a bag of transactions, e.g. the sales transactions on a given day. We consider bags $\mathcal{D}$ of d such databases, e.g. the sales transactions for different branches of store.

We assume this bag to be indexed such that by $D_i \in \mathcal{D}$ we can access the transactions sold at the i'th branch. Let $\mathcal{J} = \{1, \cdots, d\}$ be the set of indexes. An index set $j \in \mathcal{P}(\mathcal{J})$ then identifies a subset of databases $\{D_i \in \mathcal{D} \mid i \in j\}$. Finally, $U \subseteq \mathcal{P}(\mathcal{J})$ identifies those subsets of databases the user specifies as interesting.

We say that a transaction $t \in D$ supports an itemset $X \subseteq \mathcal{I}$, iff $X \subseteq t$. The support $supp_D(X)$ of X in D is the number of transactions in the database where X occurs. The relative support of X is its frequency, $freq_D(X) = supp_D(X)/|D|$, with $|D|$ for the number of transactions in D. Further, let $||D|| = \sum_{t \in D} |t|$ the total number of items. For $\mathcal{D}$, we write $|\mathcal{D}| = \sum_{D_i \in \mathcal{D}} |D_i|$, and define $||\mathcal{D}||$ analogue.

All logarithms are to base 2, and by convention we use $0 \log 0 = 0$.

3.2 MDL, a brief primer

The MDL (Minimum Description Length) [12,5] principle, like its close cousin MML (Minimum Message Length) [19,18], is a practical version of Kolmogorov complexity [6,7]. All three embrace the slogan *Induction by Compression*. For MDL, this principle can be roughly described as follows.

Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimises

$$L(M) + L(\mathcal{D} \mid M)$$

where $L(M)$ is the length, in bits, of the description of model M, and $L(\mathcal{D} \mid M)$ is the length, in bits, of the description of the data when encoded with M.

This is called two-part MDL, or *crude* MDL. As opposed to *refined* MDL, where model and data are encoded together [5]. We use two-part MDL because we are specifically interested in the model: the pattern sets that yield the best compression. Although refined MDL has stronger theoretical foundations, it can only be computed in special cases. From refined MDL we will use prequential coding to encode the data without bias. Note that MDL requires the compression to be *lossless* in order to allow for fair comparison between different $M \in \mathcal{M}$.

To use MDL in practice we have to define our model class $\mathcal{M}$, how to describe a model $M \in \mathcal{M}$, and how a model M describes the data $\mathcal{D}$. In MDL we are only interested in the length of the description, and never in the encoded data. That is, we are only concerned with the *length* of the encoding, not with materialised codes.

4 MDL for the Difference and the Norm

We first informally introduce our problem, and then formalise our objective.

4.1 The Problem, informally

Suppose we are given a bag $\mathcal{D}$ of transaction databases. Loosely speaking, by MDL we are after those patterns – itemsets – that together describe these databases best. More in particular, we want to optimally jointly characterise the database subsets U that the user specified as interesting. Our model $\mathcal{S}$ will hence consist of a *set of patterns* S_j for every $j \in U$. Every individual database $D_i \in \mathcal{D}$ will be described – characterised – using the union of all $S_j \in \mathcal{S}$ associated with D_i in the sense that i is an element of j. This allows us to associate patterns with that database subset j they are most characteristic for. Not only does this makes the overall description of the databases more efficient – no duplication is necessary – it also makes the model more insightful – if a pattern is characteristic for all databases, it will be included in the pattern set that is associated with all databases, when it is characteristic only for one database it will only be included in the pattern set associated with that particular database, etc.

We will now formally introduce our objective.

4.2 Our Models

A model $\mathcal{S}$ is a set of pattern sets $S \subseteq \mathcal{P}(\mathcal{I})$, such that every $S_j \in \mathcal{S}$ is associated with one of the database subsets $j \in U$ the user identifies as interesting. To describe an individual database $D_i \in \mathcal{D}$, we consider the union of all pattern sets in $\mathcal{S}$ that are associated with D_i, and to make sure every database over $\mathcal{I}$ can be encoded without loss, we also add all singletons Formally, we write $\pi_i(\mathcal{S}) = \{S_j \in \mathcal{S} \mid i \in j\}$ for the subset of $\mathcal{S}$ relevant to D_i, and define the coding set C_i for D_i as $C_i = \mathcal{I} \cup \bigcup \pi_i(\mathcal{S})$.

4.3 Encoded length of the data

Next, we discuss how we describe data $\mathcal{D}$ given a model $\mathcal{S}$, and in particular how to calculate the encoded length $L(\mathcal{D} \mid \mathcal{S})$. We do so bottom up, starting by how to encode an individual transaction $t \in D$ given an arbitrary coding set C. We do so using a *cover* function $cover(t, C)$ that returns a set of patterns from C such that $\bigcup cover(t, C) = t$.

To encode the patterns in the cover of t, we will use optimal prefix codes. The length of an optimal prefix code is given by Shannon entropy [2], $-\log \Pr(X)$. To compute these lengths, we hence need the probability of a pattern X in the cover of the data. Let $usg_D(X, C) = |\{t \in D \mid X \in cover(t, C)\}|$ be the number of times a pattern $X \in C$ is used in the cover of D. Wherever clear from context we simply write $usg(X)$, and slightly abusing notation, we say $usg(C)$ for the sum of usages of coding set C, i.e. $usg(C) = \sum_{X \in C} usg(X)$. The probability of X is then $\Pr(X) = \frac{usg(X)}{\sum_{Y \in C} usg(Y)}$, and the length of its optimal prefix code $L(code(X) \mid C) = -\log \Pr(X)$.

More in particular, we will use a *prequential* coding scheme [5]. Prequential codes are Universal codes [13], which means they are asymptotically optimal *without* having

to know the usages in advance. That is, unlike for KRIMP [17] we do not have to make arbitrary choices for how to encode the usages in the model – choices that may incur undue bias. The idea behind prequential coding is simple: after every received code we re-calculate all probabilities over the data received so far, initialising the usages to ϵ. This means that at any stage we have a valid probability distribution and hence can send optimal prefix codes. Surprisingly, the *order* in which we transmit codes does not affect the encoded length – a sum of logarithms is the logarithm of a product, of which we can move its terms around at will.

For the encoded length of a transaction $t \in D$ we have

$$L(t \mid C) = L_{\mathbb{N}}(|t|) + \sum_{X \in cover(t,C)} L(code(X) \mid C) \quad ,$$

where we first encode the cardinality of the transaction, and then the patterns in its cover. For the cardinality, we use $L_{\mathbb{N}}$, the Universal code for integers [13] which for $n \geq 1$ is defined as $L_{\mathbb{N}}(n) = \log^*(n) + \log(c_0)$ with $\log^* = \log(n) + \log\log(n) + \ldots$. To make it a valid code it has to satisfy the Kraft inequality, and hence we set $c_0 = 2.865064$.

For the encoded length of a database D given a coding set C we then have

$$L(D \mid C) = L_{\mathbb{N}}(|D|) + \sum_{t \in D} L(t \mid C) \quad ,$$

where we encode the number of transactions in D using $L_{\mathbb{N}}$ and then each of the transactions in turn. Aggregating the lengths of all prequential prefix codes, we have

$$L(D \mid C) = \left[L_{\mathbb{N}}(|D|) + \sum_{t \in D} L_{\mathbb{N}}(|t|) \right] + \left[\log \frac{\prod_{j=0}^{usg(C)-1} (j + \epsilon|C|)}{\prod_{X \in C} \prod_{j=0}^{usg(X)-1} (j + \epsilon)} \right] \quad .$$

Note that the first two terms are constant for all models for the same data, and can hence be ignored during optimisation. The right hand term is the length of the data when encoded using prequential coding. By common convention, for $\epsilon = 0.5$ we have

$$L(D \mid C) = L_{\mathbb{N}}(|D|) + \sum_{t \in D} L_{\mathbb{N}}(|t|) + \log \Gamma(usg(C) + 0.5|C|) - \log \Gamma(0.5|C|) - \sum_{X \in C} \log\left((2usg(X) - 1)!!\right) - usg(X) \quad ,$$

where !! denotes the double factorial defined as $(2k-1)!! = \prod_{i=1}^{k}(2i-1)$, and Γ is the Gamma function, which is an extension of the factorial function to the complex plane. That is, $\Gamma(x+1) = x\Gamma(x)$, with relevant base cases $\Gamma(1) = 1$ and $\Gamma(0.5) = \sqrt{\pi}$. We refer the interested reader to the online appendix for more details on prequential coding and its computation. Finally, by encoding the number of databases in $\mathcal{D}$, and then simply encoding every individual database in order, we have

$$L(\mathcal{D} \mid \mathcal{S}) = L_{\mathbb{N}}(|\mathcal{J}|) + \sum_{D_i \in \mathcal{D}} L(D_i \mid C_i) \quad ,$$

for the encoded size of $\mathcal{D}$ given a model $\mathcal{S}$. This leaves discussing the encoding of $\mathcal{S}$.

4.4 Encoded length of the model

Let us first discuss $L(S_j)$, the encoded length of a pattern set $S_j \in \mathcal{S}$. We define

$$L(S_j) = L_{\mathbb{N}}(|S_j|) + \sum_{X \in S_j} \left(L_{\mathbb{N}}(|X|) - \sum_{x \in X} \log \mathit{freq}_{\mathcal{D}}(x) \right)$$

in which we first encode the number of patterns, then their cardinalities. Third, we transmit the elements of X using optimal prefix codes – allowing us to reconstruct patterns up to the names of the items – and do so using the marginal item probabilities over $\mathcal{D}$. By this choice a pattern X is equally expensive regardless of the datasets for which S_j is relevant. Note that we do not have to encode the pattern usages as we encode the data prequentially.

Finally, for the encoded length of a model $\mathcal{S}$ we have

$$L(\mathcal{S}) = L_{\mathbb{N}}(|\mathcal{I}|) + L_{\mathbb{N}}(||\mathcal{D}||) + \log \binom{||\mathcal{D}|| - 1}{|\mathcal{I}| - 1} + \sum_{S_j \in \mathcal{S}} L(S_j) \quad ,$$

where we encode the length of the alphabet, the number of items in the data, and then the support per item using an index over a canonically ordered enumeration of all possibilities of distributing $||\mathcal{D}||$ events over $|\mathcal{I}|$ labels. This cost is constant for the same data and can hence be ignored when optimising the model. It is necessary, however, if we want to compare different encodings or model classes.

4.5 The Problem, Formally

Combining the above, the total encoded length of data $\mathcal{D}$ and a model $\mathcal{S}$ is defined as

$$L(\mathcal{D}, \mathcal{S}) = L(\mathcal{S}) + L(\mathcal{D} \mid \mathcal{S}) \quad .$$

By MDL we are after the model that minimises the total encoded length. Formally, our problem definition is as follows.

Minimal Pattern Sets Problem *Let $\mathcal{I}$ be a set of items, $\mathcal{D}$ a bag of transaction databases over $\mathcal{I}$, U a set of index sets for $\mathcal{D}$,* cover *a cover algorithm, and $\mathcal{F}$ the space of all admissible models, $\mathcal{F} = \mathcal{P}(\mathcal{P}(\mathcal{I}))^{|U|}$. Find the set of pattern sets $\mathcal{S} \in \mathcal{F}$ with the smallest $\bigcup \mathcal{S}$ such that the corresponding total compressed size $L(\mathcal{D}, \mathcal{S})$ is minimal.*

The search space we have to consider for this problem is rather large – even if we take into account that only patterns that occur in the data can be used to describe the data. Moreover, it does not exhibit structure we can exploit to efficiently find the optimal pattern sets, such as submodularity or (weak) monotonicity.

Hence, we resort to heuristics.

5 Algorithm

To discover good models directly from data, we propose the DIFFNORM algorithm.

5.1 The COVER algorithm

First, however, we need a cover function $cover(t, C)$ to determine which patterns from C will be used to describe transaction t. Ideally $cover$ minimises $L(\mathcal{D}, \mathcal{S})$. However, as there exists a complex non-linear relation between the total encoded length and the individual usages of patterns, optimising the cover is non-trivial [15]. We therefore adopt the greedy heuristic successfully used in KRIMP [17]. That is, we greedily cover transaction t with non-overlapping patterns from C. We do so in **Standard Cover Order**, i.e. we consider the patterns in C sorted descending on cardinality, on support, and lexicographically. The intuition is that by doing so we need as few as possible, as frequent as possible patterns to cover t. Algorithm 1 gives the pseudo-code.

Algorithm 1: GREEDYCOVER

Input: A transaction t over items $\mathcal{I}$ and a coding set C
Output: A $cover(t, C) \subseteq C$

```
1 for X ∈ C in Standard Cover Order do
2     if X ⊆ t then return {X} ∪ cover(t \ X, C) ;
3 return ∅
```

5.2 The DIFFNORM algorithm

Next we discuss the DIFFNORM algorithm. We give the pseudo code as Algorithm 2. The main intuition is that we iteratively reduce redundancy in the current description of the data by adding combinations of existing patterns. That is, we take a SLIM-like approach [14]. We start with empty pattern sets (line 1). We iteratively generate candidates in the form of $X \cup Y$ with $X, Y \in \mathcal{S} \cup \mathcal{I}$. We consider these in order of estimated gain (2). (We postpone the details of ΔL to Sec. 5.4.) Note that we can easily impose additional constraints (e.g. minimum support) to accommodate user preferences.

Per candidate, we calculate the difference in bits when adding it to the coding set for each database (line 3–4). We use these gains to determine to which pattern set(s) $S_j \in \mathcal{S}$ we will add the candidate (5–7). We do so greedily (6). We first sort the user specified index sets U descending on gain, cardinality, and last lexicographically. We iteratively pick the top-most index set, and updating the gain scores of the remaining sets by removing the gain for data sets already covered by the chosen index sets, and stop when we cannot select an index set with positive gain.

As the new pattern may have superseded the use of older ones, we have to PRUNE the model [17]. We give the pseudo-code as Algorithm 3. In a nutshell, we simply iteratively re-consider every pattern in $\mathcal{S}$ for which the usage has decreased – as these are now more expensive to encode – ordered by how much the usage has decreased. After pruning we iterate until we cannot find any patterns that improve the total encoded length. Before we return the patterns, we order them by their relative importance – the number of bits we would have to spend extra if the pattern would not be included.

Algorithm 2: DIFFNORM

Input: A bag $\mathcal{D}$ of transaction databases over items $\mathcal{I}$, and a database index set U including at least the individual indices over $\mathcal{D}$

Output: An approximation of the MDL-optimal model $\mathcal{S}$ for $\mathcal{D}$

$\mathcal{S} \leftarrow \{\emptyset \mid j \in U\}$;
for $Z \in \{X \cup Y \mid X, Y \in \mathcal{S} \cup \mathcal{I}\}$ descending on $\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus Z)$ **do**
  for $D_i \in \mathcal{D}$ **do**
    $gain_i \leftarrow \Delta L(D_i \mid C_i \oplus Z)$;
  $w \leftarrow \{\Delta L(\mathcal{D}, \mathcal{S} \oplus_j Z) \mid j \in U\}$;
  $U' \leftarrow$ WEIGHTEDGREEDYCOVER($\mathcal{J}, U, w$);
  $\mathcal{S}' \leftarrow \mathcal{S}$ with Z added to every S_j with $j \in U'$;
  $\mathcal{S} \leftarrow$PRUNE($\mathcal{D}, \mathcal{S}, \mathcal{S}'$);
Order every $S_j \in \mathcal{S}$ descending on $\Delta L(\mathcal{D}, \mathcal{S} \ominus_j Z)$;
return $\mathcal{S}$;

Algorithm 3: PRUNE

Input: A bag $\mathcal{D}$ of databases over $\mathcal{I}$, a previous model $\mathcal{S}$ and a current model $\mathcal{T}$

Output: A pruned model $\mathcal{T}$

$Cands \leftarrow$ all patterns $X \in \mathcal{T}$ for which $usg(X, \mathcal{T}) < usg(X, \mathcal{S})$;
for $X \in Cands$ in **Standard Pruning Order do**
  if $L(\mathcal{D}, \mathcal{T} \ominus X) < L(\mathcal{D}, \mathcal{T})$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} \ominus X$;
    Add all patterns $Y \in \mathcal{T}$ for which usg reduced to $Cands$;
return $\mathcal{T}$;

5.3 Candidate Generation and Evaluation

The naive approach to optimising a model is to first mine all frequent patterns $\mathcal{F}$ in $\mathcal{D}$, and then iteratively consider these as candidates. KRAMP [14] is the locally optimal strategy of iteratively adding that $Z \in \mathcal{F}$ to the model that maximises compression. Being quadratic in the size of the candidate set, this approach is prohibitively costly. KRIMP considers these candidates in a fixed order, greedily selecting those that improve compression [17]. Considering every candidate only once and in a static order KRIMP is linear in the number of candidates, but quality suffers and as all candidates need to be pre-mined and ordered materialised the approach remains costly.

Instead, we can iteratively refine the current model by searching for redundancies. Translated to our setting, SLAM [14] is the locally optimal approach. It iteratively evaluates all pairwise combinations $X, Y \in \mathcal{S} \cup \mathcal{I}$, accepting that $X \cup Y$ which maximises compression. SLIM [14] considers the same candidates, but evaluates these in order of estimated quality, accepting the first that improves compression. This leads to much improved run time and overall description length close to SLAM.

Loosely speaking DIFFNORM follows the same adage as SLIM. However, unlike SLIM, we consider multiple pattern sets – each of which relevant to different set of databases. When we extend SLIM naively, we would generate overly many candidates

and evaluate them on by far too many pattern sets and databases. To refine this process we make use of the fact that MDL punishes redundancy – which means that patterns will only be included in pattern sets they are most relevant for.

First we adapt the candidate generation process. We observe that it is very unlikely that $X \cup Y$ will be used much when X and Y are drawn from pattern sets that are not used to describe the same database. This observation allows us to refine the SLIM strategy as follows. Instead of considering all pairs $X, Y \in \mathcal{S} \cup \mathcal{I}$, we consider only $X \cup Y$ if they co-occur in a coding set C for a database D. Formally, we consider only $X \cup Y$ for $X \in S_j$ and $Y \in S_k$ with $j \cap k \neq \emptyset$ as candidates.

Next, we take a closer look at the candidate evaluation process. When we consider a pattern $X \cup Y$ with X from a pattern set S_j that is more 'specific' than the pattern set S_k that we draw Y from, that is, $j \subset k$, it will be very unlikely that $X \cup Y$ will be a good candidate to add to S_k – otherwise, X would have resided in S_k. We use this intuition and in these cases only consider to add this candidate to S_j, not to S_k. More in general, we evaluate the candidate in all $S_l \in \mathcal{S}$ with $l \subseteq j$. When j and k overlap, but j is not a strict subset of k, we evaluate the candidate in all $S_l \in \mathcal{S}$ with $l \subset j$ or $l \subset k$.

5.4 Estimating Candidate Quality

As we aim to minimise the description length, the quality of a candidate Z is the gain in total compressed size when we would add Z to pattern set $S_j \in \mathcal{S}$, i.e. $\Delta L(\mathcal{D}, \mathcal{S} \oplus_j Z)$. Formally,

$$\begin{aligned}\Delta L(\mathcal{D}, \mathcal{S} \oplus_j Z) &= L(\mathcal{D}, \mathcal{S}) - L(\mathcal{D}, \mathcal{S} \oplus_j Z) \\ &= \Delta L(S_j \oplus Z) + \sum_{i \in j} \Delta L(D_i \mid C_i \oplus Z)) \\ &= L(S_j) - L(S_j \oplus Z) + \sum_{i \in j} L(D_i \mid C_i) - L(D_i \mid C_i \oplus Z))\end{aligned}$$

Note that $\Delta L(D_i \mid C_i \oplus Z)$ is constant regardless to which pattern set $S_j \in \mathcal{S}$ we add Z – as long as i is in the index set j. Calculating the actual gain for every candidate is prohibitively costly, however – we need to cover all relevant databases to re-determine the usages. Instead, we therefore estimate the gain in bits when adding a pattern Z to pattern set S_j, i.e. $\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus_j Z)$. We then use WEIGHTEDGREEDYCOVER to get the total estimated gain, $\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus Z)$, from $\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus_j Z)\ \forall\ j \in U$. To this end we assume that as candidate we consider the union of patterns $X, Y \in \mathcal{S} \cup \mathcal{I}$, and that adding $X \cup Y$ to pattern S_j will affect only the usages of X and Y and not that of other patterns in $\mathcal{S}$. Formally, we have

$$\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus_j X \cup Y) = \Delta\widehat{L}(S_j \oplus X \cup Y) + \sum_{i \in j} \Delta\widehat{L}(D_i \mid C_i \oplus X \cup Y) \quad ,$$

where for the estimated difference in encoded length of S_j we have

$$\begin{aligned}\Delta\widehat{L}(S_j \oplus X \cup Y) &= L(S_j) - L(S_j \oplus X \cup Y) \\ &= L_{\mathbb{N}}(|X \cup Y|) - \sum_{x \in X \cup Y} \log \mathit{freq}_{\mathcal{D}}(x) \quad .\end{aligned}$$

Somewhat more intimidating, for the estimated encoded length of the data we have

$$\begin{aligned}\Delta\widehat{L}(D_i \mid C_i \oplus X \cup Y) = \log(\Gamma(usg(C) + \epsilon|C|)) - \log(\Gamma(\widehat{usg}(C') + \epsilon|C'|)) + \\ \log(\Gamma(\widehat{usg}(X, C') + \epsilon)) - \log(\Gamma(usg(X, C) + \epsilon)) + \\ \log(\Gamma(\widehat{usg}(Y, C') + \epsilon)) - \log(\Gamma(usg(Y, C) + \epsilon)) + \\ \log(\Gamma(\widehat{usg}(X \cup Y, C') + \epsilon)) - \log(\Gamma(\epsilon)) + \\ \log(\Gamma(\epsilon|C'|)) - \log(\Gamma(\epsilon|C|))\end{aligned}$$

were $C' = C \cup \{X \cup Y\}$, and $\widehat{usg}(Z, C')$ is the estimation of the usage of pattern Z when covering the data using C'. We estimate the usage of $X \cup Y$ optimistically, assuming it will be used wherever X and Y were co-used. That is, we say $\widehat{usg}(X \cup Y, C') = |utids(X) \cap utids(Y)|$, where $utids(X) = \{tid(t) \mid t \in D, X \in cover(t, C)\}$ are the ids of the transactions covered using X. Following the same assumption, we have $\widehat{usg}(X, C') = usg(X, C) - \widehat{usg}(X \cup Y, C')$, and analogue for Y.

Since we only generate and evaluate the candidates against their relevant coding sets C_i, we do the same when estimating gain. Further, to avoid re-computing all estimates at every iteration we cache the estimated gains of patterns. However, whenever a candidate Z is added to or pruned from $\mathcal{S}$ the usages of other patterns $X \in \mathcal{S}$ may change – and hence so should the estimates of any candidates that use X. We re-estimate the gains of these candidates, and maintain those for the other candidates.

5.5 Complexity

Finally, we analyse the computational complexity of DIFFNORM. In worst case, a model $\mathcal{S}$ contains all the frequent patterns $\mathcal{F}$. Let $|\mathcal{S}|$ be the total number of patterns in model $\mathcal{S}$. At worst, generating the candidates takes $\mathcal{O}((|\mathcal{S}| + |\mathcal{I}|)^2) \subseteq \mathcal{O}(|\mathcal{F}|^2)$ steps. Calculating the gain takes $\mathcal{O}(|\mathcal{S}|) \subseteq \mathcal{O}(|\mathcal{F}|)$ steps. WEIGHTEDGREEDYCOVER takes $\mathcal{O}(|U| \times \log|U|)$ steps for sorting U and $\mathcal{O}(|U|^2)$ steps for greedy selection and gain re-computation. Finally, PRUNE takes $\mathcal{O}(|\mathcal{S}|^2 \times |\mathcal{D}|)$ steps. Altogether, the worst case computational complexity is $\mathcal{O}(|\mathcal{F}|^3 \times |\mathcal{D}|)$. In practice, DIFFNORM is fast. First, MDL restricts the number of patterns in the model, pruning keeps the model non-redundant, and model changes rarely affect many patterns. Second, we generate candidates not naively from $\mathcal{S}$ but over coding sets C, and evaluate candidates only on the relevant databases D_i.

6 Experiments

We implemented our algorithm in C++ and provide the source code for the research purposes, along with the used datasets, and synthetic dataset generator.[1] All experiments were executed single-threaded on Intel Xeon E5-2643 v3 machines with 256 GB memory running Linux. We report the wall-clock running times.

We consider both synthetic and real-world data. We give the basic statistics of the real-world datasets in Table 1. For each dataset we give the number of rows, size of the

[1] `http://eda.mmci.uni-saarland/diffnorm/`

Table 1. Base statistics of the datasets used in the experiments. We report the number of rows, the size of the alphabet, the total number of items, and the number of databases.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\lvert\mathcal{D}\rvert\rvert$	$\lvert\mathcal{J}\rvert$
Adult	48 842	97	726 165	2
ChessBig	28 056	58	196 392	18
Nursery	12 960	32	116 640	5
Mushroom	8 124	119	186 852	2
PageBlocks	5 473	44	60 203	5
Led7	3 200	24	25 600	6
Chess	3 196	75	118 252	2

alphabet $|\mathcal{I}|$, total number of items and number of databases. For readability we use the shorthand notation $L\% = \frac{L(\mathcal{D},\mathcal{S})}{L(\mathcal{D},\mathcal{S}_0)}\%$ for the relative compressed size of $\mathcal{D}$ with $\mathcal{S}_0$ the model consisting of only empty pattern sets – lower is better.

In all experiments we consider $U = \{\{1\}, \ldots, \{|\mathcal{D}|\}, \Omega\}$ where $\Omega = \{1, \ldots, |\mathcal{D}|\}$. That is, we want a pattern set S_i per individual $D_i \in \mathcal{D}$, and in addition we want to have a pattern set S_Ω that contains the patterns characteristic to all databases in $\mathcal{D}$.

6.1 Synthetic Data

First, we consider synthetic data to study the behaviour of DIFFNORM on data with known ground truth. We divide the possible data into four categories: data with no patterns included, data where patterns are local to individual databases, data where patterns occur globally in every database, and data where we mix global and local patterns, i.e. data containing both local and global patterns.

For each setup we generate a $\mathcal{D}$ of two databases of $5\,000$ rows each over 120 items. We randomly plant non-overlapping patterns of cardinality uniformly chosen over the range of 4 to 8, with random frequency over the range 10% to 30%. In addition, we add 5% uniform noise. We run DIFFNORM with a minimum support of 4.5%. Table 2 shows the result of DIFFNORM per synthetic dataset, i.e. number of planted patterns, the total encoded size given the simplest model $\mathcal{S}_0$, relative compressed size $L\%$. Further, following [20], we report the number of exactly recovered patterns, the number of discovered patterns that are unions or subsets of unions of planted patterns, the number of discovered patterns that correspond to intersections between planted patterns, and the number of patterns that are tainted with, or completely due to noise.

We find that for *Random*, DIFFNORM correctly infers that the data does not contain any patterns. As for the other datasets, DIFFNORM discovers exactly all the planted patterns. In addition, DIFFNORM discovers patterns that are the union, or a subset thereof, of planted patterns X and Y – this is due to generative process. As we allow multiple patterns on the same row, particularly when X and Y are very frequent their combination can also become frequent, and therewith descriptive for the data. Overall, DIFFNORM identifies all the interesting patterns from the synthetic datasets.

Table 2. DIFFNORM recovers true patterns Results on synthetic data. Per dataset we give the number of planted patterns, the baseline description length, and the relative compression $L\%$ we obtain. Further, we report the total number of patterns DIFFNORM discovers, and break this down into the number of exactly recovered patterns (=), the number of discovered patterns that are (subsets of) unions of planted patterns ($\cup$), the number of discovered patterns that are intersections of planted patterns ($\cap$), the number of patterns unrelated to planted patterns (?).

				Discovered Patterns				
Dataset	**# planted**	$L(\mathcal{D}, \mathcal{S}_0)$	$L\%$	$\lvert\mathcal{S}\rvert$	=	$\cup$	$\cap$	?
Random	0	387 201	100	0	0	0	0	0
Local	16	587 557	22.1	17	16	1	0	0
Global	10	1 180 530	22.4	17	10	7	0	0
Mixture	19	1 008 546	11.9	29	19	10	0	0

6.2 Real World Data

Next, we investigate the performance of DIFFNORM on real-world data. In particular, we first consider seven datasets from Frequent Itemset Mining Implementations (FIMI) repository.[2] For these experiments we set a minimum support of 2. The result on FIMI datasets is given in Table 3. We see that DIFFNORM is efficient, requiring only seconds for these datasets. Moreover, we find that it achieves very good compression ratios (lower is better), and returns only modest numbers of patterns.

This leaves us to compare these numbers. This is more difficult than it may seem. For starters, comparing description lengths only makes sense when we consider the same model class and exactly the same data. Comparing on the number of discovered patterns is not trivial either. Comparing to the number of (closed) frequent itemsets [11] is not fair as it is not meant to give a summary of the data. Supervised methods [21] only report patterns that set classes apart, and do not describe the data awhole. Summarisation methods such as SLIM [14] do give succinct description per database, but lack a way to identify patterns common between the databases. Considering the number of patterns discovered summed over all databases would be hugely inflated. Most fair, we find is to compare to the number of patterns discovered over the whole data, i.e. over $\mathcal{D}_\cup = \bigcup \mathcal{D}$. We run both DIFFNORM and SLIM on this database with minsup 2 and report the number of discovered patterns. We see that DIFFNORM discovers roughly the same number of patterns as before, while SLIM on the other hand generally finds many more patterns. This is likely due to overfitting as its encoding scheme does not encode pattern lengths and codes without loss.

Next, we investigate how the patterns that DIFFNORM discovers are distributed over the different databases. That is, in Figure 1 we show the sizes of the discovered pattern sets, starting with the size S_Ω, the pattern set associated with all databases, and then the sizes of each of the S_i corresponding to $D_i \in \mathcal{D}$. We see that the patterns are nicely

[2] `http://fimi.ua.ac.be/data/`

Table 3. DIFFNORM discovers succinct descriptions. Results of DIFFNORM on the real data sets. For DIFFNORM we give the baseline compression cost $L(\mathcal{D}, \mathcal{S}_0)$, the relative compressed size $L\%$ (lower is better), the wall-clock time in seconds, and the total number of discovered patterns. For comparison, in addition we report the number of patterns DIFFNORM and SLIM [14] discover when we concatenate all databases into $\mathcal{D}_\cup = \bigcup \mathcal{D}$.

	DIFFNORM ($\mathcal{D}$)				DN($\mathcal{D}_\cup$)	SLIM ($\mathcal{D}_\cup$)
Dataset	$L(\mathcal{D}, \mathcal{S}_0)$	$L\%$	*time (s)*	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}\rvert$
Adult	3 237 869	25.6	73.5	757	782	2702
ChessBig	838 358	75.3	11	899	769	1420
Nursery	471 928	58.3	6.5	294	371	308
Mushroom	1 020 100	25.8	17	442	435	1667
PageBlocks	186 765	4.3	0	26	48	105
Led7	68 034	34.2	0	80	56	194
Chess	660 914	20.6	7.5	265	264	653

distributed over $\mathcal{S}$, it is not the case that all patterns are in either the global, or just in the local pattern sets.

We proceed to evaluate how well DIFFNORM optimises our objective. As we do not know the true optimum, we look at how the relative compression $L\%$ converges over the search iterations. An iteration here refers to the event when a pattern is accepted by DIFFNORM. As shown in Figure 2(a), for the *Adult* dataset, the relative compression reduces very sharply in the beginning and after a certain number of iterations it converges more slowly as it then needs to refine the more general patterns discovered in the first iterations. Note that as we PRUNE the number of iterations and the final number of patterns differ.

DIFFNORM relies heavily on the quality of estimating ΔL. We evaluate the quality of our estimate by checking how well $\Delta\widehat{L}(\mathcal{D}, \mathcal{S} \oplus X \cup Y)$ correlates to the actual gain $\Delta L(\mathcal{D}, \mathcal{S} \oplus X \cup Y)$. We consider *Adult* and plot the estimated and actual gain for all the candidates considered by DIFFNORM in Figure 2(b). We see that the two are strongly correlated, as most of the points lie along the diagonal, particularly those for high gain candidates. This also explains the shape of the convergence curve in Figure 2(a) – candidates with higher estimated gains are tested during early stages of the algorithm. We find that for lower estimated gains the correlation is weaker. This is explained by our assumption that all usages of all patterns in $\mathcal{S}$ remain constant, except for X and Y.

In our final experiment, we evaluate DIFFNORM qualitatively. For this, we consider the *ICDM* dataset.[3] This data consists of the abstracts – stemmed and stop-word removed – of 859 papers published at ICDM. We divide the data into two classes: one of the abstracts that do contain the word *mining* (359 rows) versus the remainder (500 rows). With the minimal support set to 5, DIFFNORM takes 71.5 seconds, and discovers 637 patterns in total, 35 for the first class, 54 for the second, and 548 in S_Ω. As expected,

[3] Available from the authors of [3].

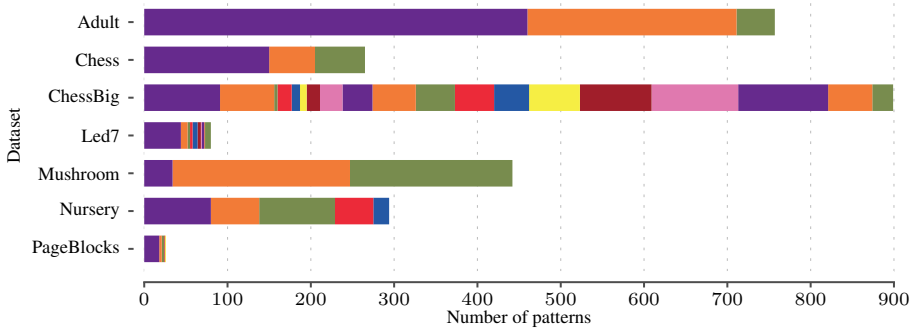


Fig. 1. DIFFNORM discovers both local and global patterns For FIMI datasets, we show the number of patterns in each pattern set discovered by DIFFNORM as indicated by the width of each colored box. The leftmost purple colored box indicates the global pattern set $\mathcal{S}_\Omega$.

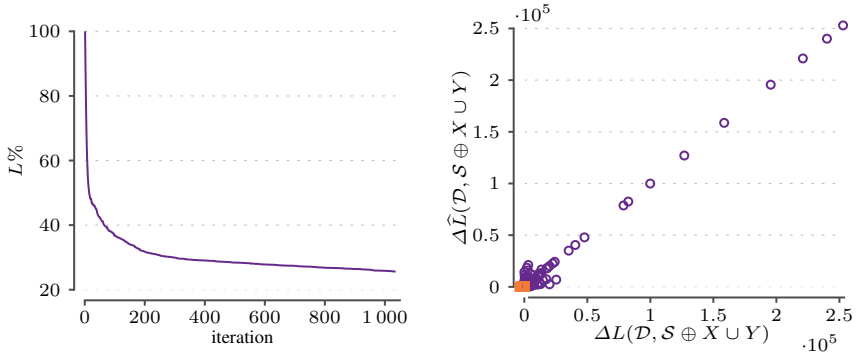
we find that the patterns found in abstracts containing *mining* point more towards exploratory analysis. The patterns discovered from abstracts not containing *mining* point more towards machine learning. On the global patterns, we find commonly used phrases in research papers like "state of the art", "evaluation", etc. We give 5 highly characteristic exemplars drawn from the top-10 of each pattern set in Table 4.

Table 4. DIFFNORM finds meaningful patterns Results of DIFFNORM on the *ICDM* dataset when split on abstracts including the word 'mining' and those that do not. Shown are five patterns per pattern set, where S_Ω is the pattern set associated with both databases.

S_{mining}	$S_{\neg\text{mining}}$	S_Ω
association rule large database	accuracy learn work	algorithm exp. problem result
fp tree	svm machine	framework general model
prune previous	cluster partition	method large set
strategy freq. pattern discover	classifier train	state [of the] art
support threshold	approach learn	evaluation technique

7 Discussion

The experiments show that DIFFNORM works well in practice. On synthetic data we recover all planted patterns exactly, and returns these top-ranked in the output. On the real world data DIFFNORM discovers succinct descriptions, returning on average less than half as many patterns as SLIM [14]. Moreover, the ICDM abstracts data show that the results are clean and easily interpretable. Finally, we showed that DIFFNORM efficiently optimises its objective score thanks to effective quality estimation of candidates.



(a) Convergence of $L(\mathcal{D}, \mathcal{S})$.

(b) Correlation of Estimation and Actual Gain.

Fig. 2. DIFFNORM searches efficiently and estimates accurately For *Adult* we show (left) the convergence of the relative compression $L\%$ per search iteration, and (right) the correlation between the estimated and actual gains of candidates. Candidates marked as circles were accepted whereas those marked as crosses were rejected.

Although these results are very encouraging, we see many possibilities to further improve DIFFNORM. A particular strong point is that our concept of multiple pattern sets and prequential coding can be extended to other data and pattern types, such as serial episodes [15]. Moreover, it will make for engaging future work to extend DIFFNORM such that it can automatically discover the optimal U for a given set of databases, and/or simultaneously find the optimal partitioning of a single given database.

Last, we have to note that MDL is not a magic wand. That is, even though we use prequential coding our objective function involves choices, and so does the optimisation. Currently we encode patterns in the pattern sets using the global singleton frequencies. In certain settings it may more sense to use the frequencies over the relevant databases instead. Extending DIFFNORM to allow for overlap would likely lead to even more succinct descriptions.

8 Conclusion

We studied how we can characterise the differences and similarities between a set of databases using pattern sets. We formalised the problem in terms of the Minimum Description Length principle [5], defining the best set of pattern sets as the one that gives the most succinct description of the data. To find good models directly from data we introduced the parameter-free DIFFNORM algorithm. Empirical evaluation showed that DIFFNORM discovers easily interpretable and non-redundant summaries that clearly identify which patterns are globally, and which ones are locally important. Future work includes refining the encoding and extending towards other data and pattern types, as well as exploring how well the patterns DIFFNORM selects perform in classification.

Acknowledgements

The authors thank the anonymous reviewers for detailed comments. Kailash Budhathoki and Jilles Vreeken are supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government.

References

1. Charu C. Aggarwal and Jiawei Han, editors. *Frequent Pattern Mining*. Springer, 2014.
2. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.
3. Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Min. Knowl. Disc.*, 23(3):407–446, 2011.
4. Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *DS*, pages 278–289, 2004.
5. Peter Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.
6. A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Peredachi Informatsii*, 1(1):3–11, 1965.
7. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.
8. Michael Mampaey, Jilles Vreeken, and Nikolaj Tatti. Summarizing data succinctly with the most informative itemsets. *ACM TKDD*, 6:1–44, 2012.
9. Pauli Miettinen. On finding joint subspace Boolean matrix factorizations. In *SDM*, pages 954–965. SIAM, 2012.
10. Siegfried Nijssen, Tias Guns, and Luc De Raedt. Correlated itemset mining in ROC space: a constraint programming approach. In *KDD*, pages 647–656. Springer, 2009.
11. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416. ACM, 1999.
12. Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.
13. Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *Annals Stat.*, 11(2):416–431, 1983.
14. Koen Smets and Jilles Vreeken. SLIM: Directly mining descriptive patterns. In *SDM*, pages 236–247. SIAM, 2012.
15. Nikolaj Tatti and Jilles Vreeken. The long and the short of it: Summarizing event sequences with serial episodes. In *KDD*. ACM, 2012.
16. Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Characterising the difference. In *KDD*, pages 765–774, 2007.
17. Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. KRIMP: Mining itemsets that compress. *Data Min. Knowl. Disc.*, 23(1):169–214, 2011.
18. Christopher S. Wallace. *Statistical and inductive inference by minimum message length*. Springer, 2005.
19. Christopher S. Wallace and D. M. Boulton. An information measure for classification. *Comput. J.*, 11(1):185–194, 1968.
20. Geoffrey Webb and Jilles Vreeken. Efficient discovery of the most interesting associations. *ACM TKDD*, 8(3):1–31, 2014.
21. Albrecht Zimmermann and Siegfried Nijssen. Supervised pattern mining and applications to classification. In Charu C. Aggarwal and Jiawei Han, editors, *Frequent Pattern Mining*, pages 425–442. Springer, 2014.

A Prequential Coding

A prefix code is a type of coding where no code is the prefix of another code. This allows the message to be transmitted as a sequence of concatenated codes without any separating symbols. The length of an optimal prefix code is given by Shannon entropy [2], $-\log \Pr(X)$. The assumption here is that the probability distribution P is static – during the transmission process $\Pr(X)$ does not change. In order to use the prefix code, we need to know the $\Pr(X)$ beforehand. That is these probabilities need to be transmitted as part of the cost of the model. And the choices on how we do this are relatively arbitrary [5].

A prequential code, on the other hand, is a prefix code defined over the predictive sequential probability distribution $P \in \mathcal{M}$, where $\mathcal{M}$ is the model class. That is the distribution P, which gave best predictions on the previously observed data, is chosen for predicting the next outcome. An important property of this being that the prequential codes are asymptotically optimal without having to know the usages in advance. Also, we do not transmit the code of the usages anymore. Rather after every received code, we update its usage and re-calculate the probabilities over the data received so far, initializing the usages to ϵ. This means that at any stage we have a valid probability distribution and hence can send optimal prefix codes.

Let us assume a sequence S of $f = |S|$ messages over alphabet set A. Let $|A| = l$. We start by initializing the count of every alphabet $a_i \in A$ to a small constant ϵ i.e. $f_0(a_i) = \epsilon$. At this point, transmitting the values $a_i \in S$ takes $-\log \frac{f_0(a_i)}{\epsilon l}$ bits. After sending j messages from the sequence S, the usage count of the alphabet a_i becomes $f_j(a_i) = \epsilon + |\{k \mid s_k = a_i \text{ with } 1 \leq k \leq j\}|$. And transmitting a_i requires $-\log \frac{f_{j-1}(a_i)}{\sum_{a_k \in A} f_{j-1}(a_k)}$ bits where $\sum_{a_k \in A} f_{j-1}(a_k) = j + \epsilon l$. From here onwards we use f_i as the count of the i^{th} alphabet after transmitting all messages in the sequence S.

Note that, we re-calculate the probability every time a message is transmitted. Putting it altogether, the code length for transmitting the sequence S, denoted $L(S)$, is given by

$$
\begin{aligned}
L(S) &= \log \frac{\prod_{j=0}^{f-1} (j + \epsilon l)}{\prod_{i=1}^{l} \prod_{j=0}^{f_i - 1} (j + \epsilon)} \\
&= \log \frac{(f - 1 + \epsilon l)(f - 2 + \epsilon l) \ldots (1 + \epsilon l)(\epsilon l)}{\prod_{i=1}^{l} (f_i - 1 + \epsilon)(f_i - 2 + \epsilon) \ldots (1 + \epsilon)(\epsilon)} \\
&= \log \frac{\Gamma(f + \epsilon l)/\Gamma(\epsilon l)}{\prod_{i=1}^{l} \Gamma(f_i + \epsilon)/\Gamma(\epsilon)} \\
&= \log \Gamma(f + \epsilon l) - \log \Gamma(\epsilon l) - \sum_{i=1}^{l} [\log\{\Gamma(f_i + \epsilon)\} - \log\{\Gamma(\epsilon)\}]
\end{aligned}
$$

A natural and often used value for ϵ is 0.5. Using $\epsilon = 0.5$, we get

$$\begin{aligned}
L(S) &= \log \Gamma(f + 0.5l) - \log \Gamma(0.5l) - \sum_{i=1}^{l} [\log\{\Gamma(f_i + 0.5)\} - \log\{\Gamma(0.5)\}] \\
&= \log \Gamma(f + 0.5l) - \log \Gamma(0.5l) - \sum_{i=1}^{l} \left[\log \frac{(2f_i - 1)!!\sqrt{\pi}}{2^{f_i}} - \log \sqrt{\pi}\right] \\
&= \log \Gamma(f + 0.5l) - \log \Gamma(0.5l) - \sum_{i=1}^{l} \left[\log\{(2f_i - 1)!!\} + \log \sqrt{\pi} - \log(2^{f_i}) - \log \sqrt{\pi}\right] \\
&= \log \Gamma(f + 0.5l) - \log \Gamma(0.5l) - \sum_{i=1}^{l} [\log\{(2f_i - 1)!!\} - f_i]
\end{aligned}$$

where $n!! = \prod_{i=0}^{k}(n - 2i)$ such that $k = \lceil n/2 \rceil - 1$ is the double factorial of n and Γ is the gamma function i.e. an extension of the factorial function over the complex plane. That is, $\Gamma(x + 1) = x\Gamma(x)$, with relevant base cases $\Gamma(1) = 1$, $\Gamma(0.5) = \sqrt{\pi}$ and $\Gamma(0.5 + n) = \frac{(2n-1)!!\sqrt{\pi}}{2^n}$. For numerical calculations, we can use Stirling's approximation for the logarithm of gamma and double factorial.