Learning CNF Theories Using MDL and Predicate Invention

Arcchit Jain1, Clément Gautrais1, Angelika Kimmig1 and Luc De Raedt1,2
1 KU Leuven, Dept. of Computer Science; Leuven.AI, B-3000 Leuven, Belgium
2 AASS, Örebro University, Sweden

Abstract
We revisit the problem of learning logical theories from examples, one of the most quintessential problems in machine learning. More specifically, we develop an approach to learn CNF-formulae from satisfiability. This is a setting in which the examples correspond to partial interpretations and an example is classified as positive when it is logically consistent with the theory. We present a novel algorithm, called Mistle - Minimal SAT Theory Learner, for learning such theories. The distinguishing features are that 1) Mistle performs predicate invention and inverse resolution, 2) is based on the MDL principle to compress the data, and 3) combines this with frequent pattern mining to find the most interesting theories. The experiments demonstrate that Mistle can learn CNF theories accurately and works well in tasks involving compression and classification.

1 Introduction
Learning propositional logical theories from examples is one of the quintessential problems in machine learning, as it is the setting used to introduce the notion of probably approximately correct learning [Valiant, 1984]. It is also the most basic setting for inductive logic programming [Muggleton and De Raedt, 1994] and logical and relational learning [De Raedt, 2008]. Furthermore, the problem of frequent pattern mining, as introduced by Agrawal et al., can also be viewed as a logical learning problem as itemsets are logical interpretations or variable assignments.

One of the most fascinating aspects of logic learning is predicate invention [Muggleton, 1987; Muggleton and Buntine, 1988; Muggleton et al., 2014], that is, the ability to automatically introduce new predicates in the language that not only compresses the data, but also provides insights into the underlying regularities. Predicate invention has been studied almost exclusively in the context of logic program induction (using Prolog) with the early approaches of Muggleton; Muggleton and Buntine relying on the Minimum Description Length (MDL) principle [Grünwald, 2007]. This form of predicate invention is also related to recent approaches for pattern and itemset mining such as KRIMP [Vreeken et al., 2011] that identify codes to compress datasets. These approaches have been widely welcomed in the data mining literature [Smets and Vreeken, 2012; Van Leeuwen and Vreeken, 2014; Fischer and Vreeken, 2019].

We revisit these approaches and combine the above three settings: 1) As Valiant, we focus on learning CNFs 2) as Muggleton, we employ inverse resolution and MDL to invent new predicates and 3) as KRIMP, we use frequent pattern mining to find the most interesting predicate definitions or codes. At the same time, we learn from satisfiability. This means that the examples are partial interpretations, that is, assignments of truth-values to some of the variables. An example e is then considered positive for a CNF theory T if T ∧ e 6= 0, that is, the theory with the example is satisfiable [De Raedt and Dehaspe, 1997]. This differs from Valiant’s setting and the typical itemset mining setting in the use of partial rather than complete interpretations (assignments to all the variables). Furthermore, it differs from Muggleton’s settings in that we learn a CNF formula rather than a logic program, which are not only more general, but also have a different semantics.

While a general framework for mining Boolean expressions has been proposed [Zhao et al., 2006], most pattern mining techniques focus on learning DNF formulae. The few approaches that focused on learning CNF theories include [Dries et al., 2009; Jabbour et al., 2013]. Dries et al. consider Valiant’s k-CNF setting in their system CNF-cc and demonstrate that k-CNF formulae can be used for a wide variety of prediction tasks but do not allow for partial interpretations nor consider predicate invention. Mining4SAT [Jabbour et al., 2013] focuses only on lossless compression and does not generalize the learned CNFs from examples.

In this paper, we introduce the Mistle algorithm for learning CNFs, which combines features from related approaches with the ability to generalize, as summarized in Table 1.

<table>
<thead>
<tr>
<th>Key features</th>
<th>KRIMP</th>
<th>Mining4SAT</th>
<th>CNF-cc</th>
<th>Mistle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learns from partial data</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Does Predicate Invention</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Generalizes a theory</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Uses the MDL Principle</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learns CNF formulae</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learns a classifier</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Mistle features with related work
2 Background

2.1 Logic

We briefly review propositional clausal logic. Propositional variables, also called predicates here, are the basic building blocks. They can be assigned the values True or False. A literal $l$ is a propositional variable $v$ or its negation $\neg v$. A disjunction $(l_1 \lor \ldots \lor l_n)$ of literals $l_i$ is a clause and a conjunction $(C_1 \land \ldots \land C_k)$ of clauses $C_j$ is a clausal theory or a CNF formula.

An interpretation is an assignment of truth values to the propositional variables. When the mapping is available only for a subset of the variables in a domain, it is called a partial interpretation. An interpretation $e$ is called a completion of a partial interpretation $e$, if it has the same variable assignments as that of $e$ and assigns truth values to all variables in the domain. An interpretation $i$ satisfies a clause $(l_1 \lor \ldots \lor l_m)$ if at least one of the $l_i$ is true in $i$, and satisfies a clausal theory if it satisfies all the clauses in that theory. A theory $T$ is satisfiable if there is an interpretations that satisfies it, notation $T \not\models \emptyset$; otherwise it is unsatisfiable, notation $T \models \emptyset$. $T$ is said to satisfy a partial interpretation, $e$, if and only if $T \land e \not\models \emptyset$. It is also useful to know that for logical formulæ $\alpha$ and $\beta$, $\alpha \models \beta$ if and only if $\alpha \land \neg \beta \not\models \emptyset$.

2.2 Learning from Partial Interpretations

Partial interpretations will be used as examples in this paper. Similar to concept-learning, we talk about theories that cover examples. More formally, a logical theory $T$ covers an example $e$ if and only if $T \models e$. Here, $e$ corresponds to a partial interpretation, that is, a conjunctive expression $l_1 \land \ldots \land l_k$ stating which literals or variable assignments evaluate to True in the partial interpretation $e$. Thus, Mistle follows the learning from satisfiability [De Raedt and Dehaspe, 1997] setting.

Notice that $T \land (l_1 \land \ldots \land l_k) \models \emptyset$ holds if and only if $T \models (\neg l_1 \lor \ldots \lor \neg l_k)$ holds. Thus learning from satisfiability also corresponds to the learning from entailment setting that is dominant in inductive logic programming. It is only that one then learns from clauses, and that the role of the positive and negative examples is reversed. Let us also remark that when learning from satisfiability, it is in principle possible to use clausal theories as examples rather than partial interpretations, which provides added expressivity. This will work with Mistle as well except that the frequent pattern mining aspect described in Section 6.3 will no longer apply.

It is also useful to note the connection to the multi-instance learning setting [De Raedt, 2008]. If $T$ does not cover a partial interpretation $e$, then it does not cover any completion of $e$. On the contrary, if $e$ satisfies $T$, then there exists at least one completion of $e$ that is satisfied by $T$.

Note that when learning from satisfiability, $T_1$ is more general than $T_2$ if $T_1 \models T_2$ [De Raedt and Dehaspe, 1997], which can be used to prune and partially order the search space.

2.3 Predicate Invention

Predicate invention is the process of automatic invention of new predicates in order to represent any regularities found in the data [Muggleton, 1987; De Raedt, 2008]. This is used in Mistle and is of great importance as it allows us to compress the theory without incurring any losses.

Example 1. Let $T = (a \lor b \lor c \lor d \lor e) \land (a \lor b \lor c \lor d \lor f) \land (a \lor b \lor e \lor d \lor g)$. Notice that all the clauses in $T$ have a shared sub-clause $a \lor b \lor c \lor d$. Thus, in order to compress $T$, we can invent a new predicate $z$ and associate it with this sub-clause. This compresses $T$ to a shorter theory, $T' = (z \lor a \lor b \lor c \lor d \lor e) \land (\neg z \lor e) \land (\neg z \lor f) \land (\neg z \lor g)$.

The transformation from $T$ to $T'$ is lossless in the sense that both theories cover exactly the same set of examples, that is, the same set of partial interpretations over the original variables (the variables in the dataset). Of course, when considering also the newly introduced predicate $z$, the space of possible interpretations becomes larger.

To understand how predicate invention is performed, it is vital to look into inverse resolution. Inverse resolution refers to inferring input clauses for resolution based upon the resolvent or the output clauses. Thus, it is an inverse of the process of resolution. In Mistle, predicate invention is performed using an inverse resolution operator called $W$-operator, which is delineated later in Section 4.

In the above example, the sub-clause, $a \lor b \lor c \lor d$ can also be thought of as a pattern occurring frequently in the theory. This consequently draws parallels to frequent itemset mining. But predicate invention offers a logical approach to find regularities in the theory. Early predicate invention systems like DUCE [Muggleton, 1987] and CIGOL [Muggleton and Buntine, 1988] even interacted with the user and allowed the user to name the newly invented predicate, keeping the theory easy to understand at all stages of the algorithm.

2.4 MDL Principle

For concept learning, one could simply construct a naïve, uncompressed theory that is consistent with the data. However, as the size of such a theory explodes for larger amounts of data, compressing the theory is crucial. The MDL Principle provides a formal framework for this.

Let $D$ be the input data, $T$ be a theory for it, and $DL$ be a description length measure. Then, the MDL principle states that the best theory is the one that minimises the joint description length of the theory and the data, given by

$$DL(D, T) = DL(T) + DL(D|T)$$

The first term denotes the description length of the theory $T$, the second that of the data when it is encoded using $T$. This definition of MDL is called the two-part MDL as it separately encodes the theory and the data [Grünwald, 2007]. We define the measure $DL$ used by Mistle in Section 5.

3 Problem Description

By now, we are able to formalize the problem:

Given:

- a set of variables, $V = \{v_1, v_2, \ldots, v_m\}$
- a set of positive partial interpretations $P$
- a set of negative partial interpretations $N$
- a measure of description length $DL$
Find: a theory $T^*$ over $V$ that minimizes $DL(P \cup N, T)$,

$$T^* = \arg \min_T [DL(T) + DL(P|T) + DL(N|T)]$$

(2)

Thus we are looking for a theory that maximally compresses the data and classifies the examples as correctly as possible.

The first term in the right hand side aims for a simple theory. The second and third terms aim for fewer errors made by a theory in explaining the positive and negative partial interpretations, respectively. In particular, $DL(P|T)$ measures the encoding length of the positive examples that are not satisfied in $T$, and $DL(N|T)$ that of the negative examples that are satisfied in $T$. These terms are formally defined as:

$$DL(P|T) = DL(\{ p \mid p \in P, T \land p \models \Box \})$$

(3)

$$DL(N|T) = DL(\{ n \mid n \in N, T \land n \not\models \Box \})$$

(4)

*Mistle* addresses the above stated problem using 2 simple steps: 1) It initializes a theory that explains the given data. 2) It uses a set of compression operators based on inverse resolution to compress that theory as much as possible. We describe how the theory is initialized in Section 6.1 and delineate the compression operators used in the next section.

4 Compression Operators

We use four logical operators for compression: W-operator, V-operator, subsumption operator, and truncation operator. These operators are adapted from the inverse resolution operators in *DUCE* [Muggleton, 1987].

We use the following notation to define the operators: predicates are denoted in lowercase, like $a, z$, and clauses are denoted in uppercase, like $A, B, \text{or } C$. Furthermore, $|B|$ denotes the number of literals in $B$. The invented predicate is denoted by $z$. We define Delta ($\Delta$) as the difference in the coverage of the input and output sets:

$$\Delta = (\text{Input} \land \neg \text{Output}) \lor (\neg \text{Input} \land \text{Output})$$

Operations where $\Delta$ is not satisfiable, i.e. $\Delta \models \Box$, are called lossless operations.

**W-operator** Given two clauses $A \lor B$ and $B \lor C$, which share $B$, the W-operator compresses the input by inventing a new predicate $z$ and replacing the input with the output clauses: $A \lor \neg z, z \lor B,$ and $C \lor \neg z$. The intuition behind inventing $z$ is that $z$ can indicate whether $B$ is True or False. It is easy to see that W-operator is an inverse resolution operator as the input clauses can be inferred back from the output clauses by resolving on $\neg z$. In general,

**Input** : $(B \lor C_1) \land \cdots \land (B \lor C_{n-1}) \land (B \lor C_n)$

**Output** : $(z \lor B) \land (\neg z \lor C_1) \land \cdots \land (\neg z \lor C_n)$

**$\Delta$** : $\Box$

**V-operator** Given two clauses $a \lor B$ and $B \lor C$, that share $B$, the V-operator compresses the input by replacing them with the output clauses: $a \lor B,$ and $\neg a \lor C$. In the second input clause, $B$ is replaced by a single literal $\neg a$ leading to compression. Similar to the W-operator, the V-operator is also an inverse resolution operator because $B \lor C$ can be obtained back from the output clauses if resolved upon $a$. But unlike the W-operator, the V-operator is a generalization as $a \land B \land \neg C$ represents all the cases that are covered by the input but are not covered by the output. In general,

**Input** : $(a \lor B) \land (B \lor C_1) \land \cdots \land (B \lor C_{n-1})$

**Output** : $(a \lor B) \land (\neg a \lor C_1) \land \cdots \land (\neg a \lor C_{n-1})$

**$\Delta$** : $a \land B \land \neg (C_1 \lor C_2 \land \cdots \land C_{n-1})$

**Subsumption Operator** The Subsumption or S-operator, is a trivial operator that discards all the specific clauses that are subsumed by another clause. For example, given two clauses, $A \lor B$ and $B \lor C$, in the input, the S-operator would only return $B$ (follows from $(A \lor B) \land B \iff B$). It is a lossless operator and results in significant compression.

**Input** : $B \land (B \lor C_1) \land \cdots \land (B \lor C_{n-1})$

**Output** : $B$

**$\Delta$** : $\Box$

**Truncation Operator** The Truncation or T-operator, truncates the input clauses by retaining the shared sub-clause. For example, if there are 2 clauses, $A \lor B$ and $B \lor C$, in the input, the T-operator would return the one clause, $B$. It is clearly a generalization and should only be applied when it decreases the description length, $DL(D, T)$.

**Input** : $(B \lor C_1) \land \cdots \land (B \lor C_n)$

**Output** : $B$

**$\Delta$** : $\neg B \land C_1 \land C_2 \land \cdots \land C_n$

5 MDL Encoding in Mistle

To use the MDL principle, we need to define how a theory is encoded in *Mistle*. Encoding a theory means that the full theory of $n$ clauses, $T = C_1 \land \cdots \land C_n$ containing $m$ variables, we measure the description length of $T$ in bits (based on Grünwald 2007):

$$DL(T) = L_N(m) + L_N(n)$$

$$+ \sum_{i=1}^{n} \left( \log_2(m) + \log_2(2m - 2(j - 1)) \right)$$

(5)

where $L_N$ is the universal code for integers$^1$.

$L_N(m)$ and $L_N(n)$ are the lengths of the encodings of the number $m$ of variables and the number $n$ of clauses, respectively. The sum adds, for each clause, the length $\log_2(m)$ of the encoding of the clause length, and the length of encoding all its literals one by one (last term).

To encode the literals, we could have used a naïve encoding of $\sum_{j=1}^{[C_i]} \log_2(2m)$, as there can be $2m$ literals in the clause (the variables and their negations). Considering previously encoded literals, we exploit the fact that if $a \in C_i$, then $\neg a \notin C_i$ (otherwise the clause is trivially satisfiable), and that no literal can occur more than once in a clause. This removes two possibilities of literals after sending each literal and results in a total encoding length of $\sum_{j=1}^{[C_i]} \log_2(2m - 2(j - 1))$.

$^1L_N(x) = \log_2(2.865064) + \log_2(x) + \log_2(\log_2(x)) + \cdots$ (Referred from Equation 3.12 in [Grünwald, 2007]).
The error terms $DL(P|T)$ and $DL(N|T)$ are encoded in the same way as $DL(T)$. Table 2 provides an example.

### Algorithm 1 Mistle

**Input**: $D = \{P, N\}$ - (Sets of partial interpretations)

**Output**: A minimal CNF theory, $T$, for input data, $D$.

1: $T = \neg(\bigvee_{n \in N} n)$
2: while $T' \text{ True}$ do
3: $T' = \text{compress}(T)$
4: if $DL(D, \text{resolve}(T')) < DL(D, T')$
   then $T' = \text{resolve}(T')$
5: if $T' == T$ then break else $T = T'$
6: end while
7: $\text{prune}(T)$
8: return $T$

6.2 Compressing the Theory - $\text{compress}(T)$

Compression, described in Algorithm 2, is the most important component of Mistle. It starts with searching for a set of clauses, that contain a shared sub-clause, i.e., all the clauses in $C \subseteq T$ that share a common set of literals. The set of shared sub-clauses is stored in $F$ [line 1]. This step is performed using frequent itemset mining, and is explained in Section 6.3. For each $C$ [line 3], Mistle selects an operation that decreases $DL(D, T)$ the most w.r.t the data $D = \{P, N\}$ [line 5].

6.3 Mining Frequent Itemsets - $\text{mine}(T)$

To compress the theory $T$, Mistle needs to identify a set of clauses in $T$ that it could compress using the compression operators. This is done by mining frequent itemset mining, using the $DC1\_Closed$ algorithm [Lucchesi et al., 2004]. A theory can be considered as a set of clauses and a clause can be represented as a set of literals. For a frequent itemset mining system, the input is the theory, $T$. The transactions are the clauses that occur in $T$. The items and itemsets are the literals and sub-clauses, respectively. The idea is to mine frequent itemsets over all the clauses in $T$. Let each itemset be a sub-clause of $l$ literals and be shared by a set of $m$ clauses in $T$. Then once all the itemsets are mined, these itemsets are sorted in decreasing order based on the following criteria:

1. $(l - 1)(m - 1) - 2$: this is the literal reduction achieved by applying the $W$-operator. It is chosen as a heuristic for the possible description length reduction that can be achieved. The $W$-operator is chosen as it is always applicable on a mined itemset.
2. $l$, the length of the itemset
3. $m$, the frequency of the itemset

6.4 Resolving the Theory - $\text{resolve}(T)$

In Algorithm 2, lines 5 and 6 choose the operator that decreases the description length the most. While this guarantees that each step can only decrease the description length, applying many operations might actually make a previous operation compress less. In our case, a $T$-operation can affect general. It keeps the search process simple, systematic and efficient. So, Mistle does not require compression operators that specialize and can only use those that generalize.
the compression of an early \( W \)-operation. Recall that the \( W \)-operator can decrease the description length if the invented predicate \( z \) is used in many clauses. But, if a \( T \)-operation deletes a clause in which a predicate, invented by a previous \( W \)-operation, was used, the compression of that \( W \)-operation decreases. Eventually, this can lead to a negative compression of this \( W \)-operation; it actually increased the description length. So, we need to reverse this \( W \)-operation by resolving the output clauses upon the newly invented predicate \( z \).

This resolving step is executed in the following manner: 1) While executing a \( T \)-operation in line 3 of Algorithm 1, all the invented predicates that were contained in the clauses deleted by \( T \)-operation are cached. 2) In line 4, for every such invented predicate, the \( W \)-operation that invented that predicate is resolved back to see if the description length decreases further. 3) In case if the description length decreases, then that \( W \)-operation is reversed in the theory.

6.5 Pruning the Theory - \( \text{prune}(T) \)

We prune the learned theory to remove any clauses whose removal decrease the description length. This pruning operation removes clauses that tend to over-fit the data and unnecessarily increase the description length. Pruning is applied as a post-processing step in Algorithm 1 and not as an operator in Algorithm 2 because of its computational cost. Indeed, the pruning operation loops over all clauses of the theory, deletes them and computes the new description length. Computing this length is costly as it requires checking whether the examples are satisfiable. Pruning as a post-processing step has also been used in other MDL-based approaches such as KRIMP.

7 Experiments

Our experimental evaluation compares Mistle against the related approaches KRIMP [Vreeken et al., 2011], Mining4SAT [Jabbour et al., 2013] and CNF-cc5 [Dries et al., 2009]. As Mining4SAT and KRIMP only use lossless operations, we consider two versions of Mistle: Mistle-Lossy is the full version of the algorithm, while Mistle-Lossy only uses lossless operations, i.e., operations in line 5 of Algorithm 2 are considered only if their \( \Delta \) is unsatisfiable. We consider the tasks of compression and classification, and the following questions:

7.1 How Accurately Can Mistle Learn a Theory?

When the actual theory is known in a domain, the accuracy can be simply defined as the percentage of all complete interpretations that are either satisfied by both theories, or not satisfied by both the theories, i.e.:

\[
\text{Accuracy}(T, T^*) = \frac{1}{2^m} \left| \{\omega \mid \{(T \land \omega) \models \Box\} \land (T^* \land \omega) \models \Box\} \cup \{(T \land \omega) \not\models \Box\} \land (T^* \land \omega) \not\models \Box\} \} \right|
\]

where \( T \) and \( T^* \) are the learned and actual theories over \( m \) variables, respectively and \( \omega \) belongs to the set of all possible \( 2^m \) interpretations. We use artificially generated actual theories and randomly generate data based on those theories to test the accuracy \(^2\). To randomly generate the actual theory \( T^* \) of \( m \) variables: 1) we first randomly select the number of its clauses uniformly between 2 to \( \min(10, m) \). 2) For each clause \( c_i \), we choose \( k_i \) uniformly between 1 to \( m \) as the number of its literals. 3) Lastly, we randomly sample \( k_i \) variables without replacement and assign them a negative sign with the probability of 0.5. We then generate a dataset of \( P \) (positives) and \( N \) (negatives) based on whether a randomly sampled partial interpretation is satisfied by \( T^* \). The number of clauses in the actual theory is restricted to 10 so we could sample sufficient size of \( P \) and \( N \). To account for the cases where the size of \( P \) and \( N \) may not be balanced, Mistle learns 2 theories: \( T_+ = \text{Mistle}(P, N) \) and \( T_- = \text{Mistle}(N, P) \) and selects the one which has the minimum description length.

Figure 1: Experiment 1: Mistle accurately learns actual theories.

Figure 1 plots the mean accuracy for 100 generated theories and the shaded area represents the standard deviation around the mean. We individually vary 3 parameters: the number of examples, missingness parameter, and the number of variables, which have the default values of 400, 0.1 and 14 respectively. The missingness parameter is defined as the probability of not observing a literal in the data. For example, a missingness parameter of 0.1 implies that each literal in the data goes missing independently with a probability of 0.1.

The default parameters are chosen to test Mistle in a challenging and interesting setting. Using 400 examples corresponds to 2.5% of all the \( 2^{14} \) possibilities. A missingness parameter of 0.1 tests Mistle in a non-trivial amount of noise.

It is interesting to note that in the middle plot, Mistle learns more accurate theories when the missingness is high. This is expected, as a negative partial interpretation provides more information than a complete interpretation: all completions of a negative partial interpretation are also negative. Overall, the results of this experiment validate that Mistle performs well by learning a close to perfect CNF theory in many cases.

7.2 How Much Can Mistle Compress?

We compare Mistle against both the systems that perform compression - Mining4SAT and KRIMP. For this experiment, we use the UCI datasets [Coenen, 2003; Dua and Graff, 2017]. If a dataset has more than two classes, the most frequent class is chosen to be the positive class and the remaining classes are binned together as the negative class. Since Mining4SAT inputs a CNF theory and compresses it using operators similar to the \( W \) and \( S \) operators, we convert \( P \) and \( N \) into their respective CNFs based on Equation 6 to input to Mining4SAT: \( T_P = \neg \left( \bigvee_{p \in P} \right) \) and \( T_N = \neg \left( \bigvee_{n \in N} \right) \). KRIMP, on the other hand, is capable of directly taking the data as an input and compressing it. The minimum support threshold used for KRIMP and Mistle is chosen in such a way

\(^2\)Because the actual theory is usually unknown in real data.
that it mines at least 10000 patterns, when so many frequent patterns exist. The compression measure used here is similar to the one used by Jabbour et al. and Vreeken et al.:

\[
\text{Compression}(D, T) = 1 - \frac{DL(D, T)}{DL(D)}
\]

The results, plotted in Table 3, conclude that Mistle, even in the lossless case, compresses much more than Mining4SAT and KRIMP as it uses a wider set of compression operators.

As the theories we obtained using CNF-cc5 [Dries et al., 2009] were typically at least two orders of magnitude larger than those of Mistle and did not compress the data, we do not provide further details on this approach.

7.3 How Well Can Mistle Classify?

To test whether Mistle’s theories are not only highly compressed but also accurate, we compare Mistle with KRIMP on a classification task, using 10 fold cross-validation on the UCI datasets. Mistle learns both \( T_+ \) and \( T_- \) and then uses the theory with the minimum description length w.r.t the data.

Table 3 shows that Mistle is not as accurate as KRIMP. The difference is largely due to the two datasets, Pima and CylBands. Without these two data-sets Mistle scores a mean accuracy of 85.04\%, which is competitive with state-of-the-art pattern mining techniques to perform classification. Note that, on the TicTacToe dataset, Mistle achieves 100\% accuracy as it learns all the 8 ways (representing it in 8 clauses of 3 literals each) in which a game can be won. This experiment shows that the additional compression that Mistle obtains may come – for some datasets – with a slightly increased error rate. However, the much more compressed theories are more explainable and provide more actionable insights. Whether this is important or not will depend on the application.

Implementation Mistle is written in Python 3.7 and uses SPMF library [Fournier-Viger et al., 2016] for closed frequent itemset mining and PicoSAT library [Biere, 2008] for SAT solving. Experiment 1 is run on Intel i5 (2.3GHz; 16Gb RAM) while experiments 2 and 3 are run on Intel i7 (3.4GHz; 16Gb RAM). For Experiment 2, the average runtime of Mining4SAT, KRIMP, Mistle-Lossless, and Mistle-Lossy is 5, 7, 28, and 5 seconds respectively. For experiment 3, the average runtime of KRIMP, Mistle-Lossless, and Mistle-Lossy is 19, 162, and 48 seconds respectively. On average, Mistle is slightly slower than KRIMP, which is implemented in C++.

8 Conclusions

We introduced the Mistle algorithm for learning CNFs, whose key features are denoted in the Table 1. The experiments using both real and artificial datasets cover a wide range of tasks and compare with related works. Mistle is best at learning small human-understandable logical theories in real domains. Our experiments demonstrate that Mistle can learn CNF theories quite accurately and learns highly compressed theories.

More generally, we show that learning CNF theories is better with regard to compression that the state-of-the-art system (KRIMP) for mining sets of itemsets which are similar to the DNF mining techniques. We hope that this work can lead to further research in the domain of CNF theory learning and predicate invention, and can be extended into first order logic.

Acknowledgements

This work has received funding from the European Research Council under the European Union’s Horizon 2020 research and innovation programme (\#694980 SYNTH: Synthesising Inductive Data Models) and from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme. Luc is also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Minimum Support Threshold</th>
<th>Exp 2: Compression</th>
<th>Exp 3: Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mining4SAT</td>
<td>KRIMP</td>
</tr>
<tr>
<td>Iris-17</td>
<td>1</td>
<td>7.8</td>
<td>53.2</td>
</tr>
<tr>
<td>Iris-18</td>
<td>1</td>
<td>6.7</td>
<td>50.4</td>
</tr>
<tr>
<td>Iris-19</td>
<td>1</td>
<td>6.7</td>
<td>50.6</td>
</tr>
<tr>
<td>Glass</td>
<td>1</td>
<td>8.8</td>
<td>28.9</td>
</tr>
<tr>
<td>Wine</td>
<td>2</td>
<td>10.7</td>
<td>21.6</td>
</tr>
<tr>
<td>Ecoli</td>
<td>1</td>
<td>37.1</td>
<td>59.9</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>38</td>
<td>35.6</td>
<td>36.5</td>
</tr>
<tr>
<td>Heart</td>
<td>9</td>
<td>31.4</td>
<td>42.3</td>
</tr>
<tr>
<td>Dermatology</td>
<td>2</td>
<td>20.9</td>
<td>45.9</td>
</tr>
<tr>
<td>Auto</td>
<td>13</td>
<td>27.0</td>
<td>23.4</td>
</tr>
<tr>
<td>HorseColic</td>
<td>12</td>
<td>17.3</td>
<td>27.0</td>
</tr>
<tr>
<td>Pima</td>
<td>1</td>
<td>44.2</td>
<td>65.0</td>
</tr>
<tr>
<td>TicTacToe</td>
<td>7</td>
<td>5.4</td>
<td>37.0</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>88</td>
<td>22.7</td>
<td>17.2</td>
</tr>
<tr>
<td>Flare</td>
<td>1</td>
<td>38.7</td>
<td>61.9</td>
</tr>
<tr>
<td>CylBands</td>
<td>211</td>
<td>20.0</td>
<td>26.3</td>
</tr>
<tr>
<td>Led</td>
<td>1</td>
<td>42.8</td>
<td>75.8</td>
</tr>
</tbody>
</table>

Table 3: Performance of Mistle over compression and classification (Compression and accuracy are in %)
References


