Surprising Pattern Mining in Hypergraph as a Form of Reasoning

Nil Geisweiller[1 [0000−1111−2222−3333]] and Ben Goertzel[2,3 [1111−2222−3333−4444]]

SingularityNET

Abstract. In this paper we introduce a pattern miner algorithm alongside a definition of surprisingness. The algorithm is framed as reasoning process, implemented on top the Unified Rule Engine, OpenCog's rewriting system for carrying mathematical and common sense reasoning. Some initial experiments are conducted on SUMO ontology.

Keywords: Pattern Miner · Surprisingness · Reasoning.

1 Introduction

Finding patterns in data can be useful.

By placing more learning into reasoning it allows to

1. Leverage the existing inference control mechanism of OpenCog (to be described in subsequent publications, but that follows the abstract principles described in [REF: growth of pattern, etc].
2. Offers more transparency, both for the human and the machine, bringing a form of XAI (eXplanatory AI).

Although such a reframing carries a computational cost it confers a significant benefit, which is that the search over the space of patterns, by far the most costly aspect of pattern mining is amenable to meta-learning [REF].

Pattern mining itself, if considered in its broadest form can be a very expensive process. Implementing it on top of the Unified Rule Engine confer multiple advantages.

2 Mining Surprising Patterns in Hypergraph Data Base

2.1 Patterns as Queries

In OpenCog, data can be retrieved from the atomspace via programs describing queries, similar in spirit to SQL queries, represented themselves as hypergraphs! This insure complete reflexivity of the querying language, queries can be queried, produced, inferred, etc.

What the pattern miner does is essentially to evolve such query programs, which is underneath also an evolution of inferences.

Here's an example of such query programs (provided in Scheme syntax, the primary binding language for OpenCog)

```
(Get
  (Present
    (Inheritance (Variable "$X") (Variable "$Y"))
    (Inheritance (Variable "$Y") (Variable "$Z"))))
```

which fetch any instance of transitivity in the AtomSpace. For instance if the AtomSpace contains

```
(Inheritance (Concept "cat") (Concept "mammal"))
(Inheritance (Concept "mammal") (Concept "animal"))
(Inheritance (Concept "square") (Concept "geometric-shape"))
```

it will retrieve

```
(Set
  (List (Concept "cat") (Concept "mammal") (Concept "animal")))
```

where `cat`, `mammal` and `animal` are associated to variable `$X`, `$Y` and `$Z` according to a prefix order. `square` and `geometric-shape` are not retrieved because they do not exhibit transitivity in the data.

The construct `Get` means retrieve. The construct `Present` means that the following expressions are patterns to be conjunctively matched against the data present in the atomspace. We also call the argument expressions of `Present` clauses and say that the whole pattern is a conjunction of clauses.

The component that takes a query and returns the matching data is called the *pattern matcher*. The pattern miner solves the inverse problem of pattern matcher. That is provided data, it attempts to find queries that would retrieve a certain minimum number of matches. This number corresponds to the *support* in the pattern mining terminology.

More constructs exist to build sophisticated queries such as type declarations, preconditions, and pretty much any computation that one may need to express. However the pattern miner only supports a subset of these, due to the inherent complexity of dealing with such expressiveness. The type of queries like the example above are supported and already allows to capture a wide range of patterns. For instance, as the reader would have probably noticed, clauses can be connected via joint variables. In the example above such a joint variable is `$Y`.

2.2 Pattern Mining High Level Algorithm

Frequent Subtree Mining [2] refers to a group techniques to discover patterns within a data base of trees.

Patterns are Atomese programs, programs expressing queries, akin to SQL queries.

To discover surprising patterns we first mine frequent patterns, pruning the search using the apriori property [1], as well as some home brewed heuristics. Then we calculate the surprisingness of all resulting patterns.

Much of the terminology used in this paper is borrowed from . Let us recall the most important terms

- *Text tree*: a tree, here a hypergraph, that is part of the data set to mine. Can be simply called *tree*. Generally speaking any atom of an atomspace.
- *Pattern tree*: a tree representing a pattern, that is capturing a collection of text trees. Can be simply called *pattern*.
- *Count*: number of text trees and subtrees matching a given pattern.
- *Frequency*: count of a pattern divided by the size of the universe, here the number of nodes of the hypergraph.
- *Support*: similar to count.
- *Minimum support*: parameter of the mining algorithm to discard patterns with frequency below that value.
- *Apriori property*: assumption that allows to systematically prune the search space. In its least abstract form, it expresses the fact that if a pattern tree has a certain frequency 'f' then a specialization of it can only have a frequency that is equal to or lower than 'f'.

Pattern mining operates by searching the space of patterns, typically starting from the most abstract pattern, the *top* pattern that encompasses all trees, then constructing specializations of it, retain those that have enough support (frequency equal to or above the threshold), recursively specializing those, and so on.

Given a data base $\mathcal{D}$, a minimal support S and an initialize collection $\mathcal{C}$ of patterns with their counts equal to or above S, the mining algorithm is as follows

1. select a pattern P from $\mathcal{C}$,
2. find all shallow specializations $\mathcal{E}$ of P with support equal to or above S,
3. remove P from $\mathcal{C}$ and add the new patterns $\mathcal{E}$ to $\mathcal{C}$,
4. repeat till a termination criterion is met (maximum number of iterations, etc).

The pattern collection $\mathcal{C}$ is usually initialized with the top pattern, from which all subsequent pattern derive from.

3 Framing Pattern Mining as Reasoning

The tricky part of the algorithm above is step 1, selecting which pattern to expand at each iteration, which will determine how the space of patterns is being searched. However by reframing pattern mining as reasoning we can.

We will just mention the fact that due to having the search framed as a reasoning process, the following step actually amounts to selecting a source in a forward chaining reasoning.

The type of propositions we need to prove have the following forms

$$S \leq \text{support}(P, \mathcal{D})$$

expressing that P has enough support with respect to the data base $\mathcal{D}$, or simply

$$\text{minsup}(P, S, \mathcal{D})$$

where minsup is defined as

$$\text{minsup}(P, S, \mathcal{D}) := S \leq \text{support}(P, \mathcal{D})$$

The type of knowledge above can be inferred with the rule

$$\text{minsup}(Q, S, \mathcal{D}) \wedge \text{spec}(Q, P) \vdash \text{minsup}(P, S, \mathcal{D})$$

expressing that if Q has enough support, and Q is a specialization of P, then P has enough support, essentially formalizing the Apriori property[1].

Other rules can be used to infer knowledge about minsup, more considered as heuristics as, unlike the apriori property, they do not guaranty completeness, but can speed-up the search. For instance the following rule

$$\text{minsup}(P, S, \mathcal{D}) \wedge \text{minsup}(Q, S, \mathcal{D}) \wedge \mathcal{P}(P \otimes Q) \vdash \text{minsup}(P \otimes Q, S, \mathcal{D})$$

expresses that if P and Q have enough support, and a certain combination of P and Q has a certain property, then such combination has enough support. An example of such rule will be given below.

4 Surprisingness

Even with the help of the apriori property and additional heuristics to prune the search, the volume of found patterns can still be overwhelming. For that it can be helpful to assign to the patterns a measure of *interestingness*. Such notion is broad and can mean various things in various contexts. We will restrict our attention on the sub-notion of *surprisingness*. Although still broad one can define surprisingness as what is *contrary to expectations*. It is not too difficult to see, in the context of discovering patterns, why such notion is important. If some pattern contradicts expectations it means we have uncovered an incorrectness or incompleteness in our model of the world and thus requires special attention.

Just like for pattern mining, surprisingness can be reframed as reasoning. They are many ways to formalize such notion. We suggest that in its most general sense, surpisingness may be the difference of outcome between different inferences over the same conjecture. Of course in most logical systems, if consistent, different inferences will produce the same result. However para-consistent systems, such as PLN [5] for *Probabilistic Logic Network*, OpenCog's logic for common sense reasoning, allow different inferences to produce different outcomes on the a given conjecture. In particular PLN allows a conjecture to be believed with various degrees of truth, ranging from total ignorance to absolute certainty. Thus PLN is particularly well suited for such definition of surprisingness. More specifically we define surprisingness as the *distance of truth values between different inferences over the same conjecture*. In PLN a truth value is a second order distribution, probabilities over probabilities, as explained in Chapter 4 of [5]. Second order distributions allow to capture uncertainties. Total ignorance is

[1] or rather its simplest variation

represented by a flat distribution (Bayesian prior), or possible a slightly concave one (Jeffreys prior [6]), and absolute certainty by a Dirac delta function.

Such definition of surprisingness has the merit of encompassing a wide variety of cases; like the surprisingness of finding a proof contradicting human intuition. For instance the outcome of Euclid's proof of the infinity of prime numbers might contradict the intuition of a beginner upon observation that prime numbers rapidly rarefy as numbers grow. It also encompasses the surprisingness of observing an unexpected event, or the surprisingness of discovering a pattern in seemingly random data. All these cases can be reframed as ways of constructing different types of inferences and finding contradictions between them. For instance in the case of discovering a pattern in a data set, one inference could calculate the empirical probability based on the data, while an other inference could calculate a probability estimate based on independence of the variables and their marginal probabilities.

To be closer to the human notion of surprisingness one may refine this definition by having one type of inference as being the current model of the world from which rapid (and usually uncertain) conclusions can be derived, and the other type of inference provided by the world itself, either via observations, in the case of an experienced reality, or via crisp and long chains of deductions in the case of a mathematical reality.

The distance measure to use to compare conjecture outcomes remains to be defined. Since our truth values are distributions the Jensen-Shannon distance [4], also suggested as surprisingness measure in [7] and [3], could be used. The advantage of such distance is that it accounts well for uncertainty. If for instance a pattern is discovered in a small data set displaying high levels of dependencies between variables (thus surprising relative to an independence assumption of these variables), the surprisingness measure should consider the possibility that it might be a fluke since the data set is small. Fortunately, the smaller the data set, the flatter the second order distributions representing the empirical and the estimated truth values of the pattern, automatically reducing the Jensen-Shannon distance.

Likewise one can imagine the following coin tossing experiments. In the first experiment a coin is tossed 3 times, a probability p_1 of head is calculated, then the coin is tossed 3 more times, a second probability p_2 of head is calculated. p_1 and p_2 might be very different, but it should not be surprising given the low number of observations. On the contrary, in the second experiment the coin is tossed a billion times, p_1 is calculated, then another billion times and p_2 is calculated. Here even tiny differences between p_1 and p_2 should be considered surprising. In both cases, by representing p_1 and p_2 as second order distributions rather than mere probabilities, the Jensen-Shannon distance properly accounts for the uncertainty.

4.1 Independence-based Surprisingness

Here we consider a simple form of surprisingness based on the independence of the clauses within a pattern. For instance

```
(Get
  (Present
    (Inheritance (Variable "$X") (Variable "$Y"))
    (Inheritance (Variable "$Y") (Variable "$Z"))))
```

has two clauses

```
(Inheritance (Variable "$X") (Variable "$Y"))
```

and

```
(Inheritance (Variable "$Y") (Variable "$Z"))
```

joint by `$Y`. The goal is to, given the empirical probability of each clause, calculate the probability estimate of their conjunctions. Of course the presence of joint variables makes that a simple scalar product of the probabilities of the clauses not be adequate. To accommodate for that we use the fact that a pattern of connected clauses is equivalent to a pattern of disconnected clauses with explicit equivality between the variables. For instance

```
(Get
  (Present
    (Inheritance (Variable "$X") (Variable "$Y"))
    (Inheritance (Variable "$Y") (Variable "$Z"))))
```

is equivalent to

```
(Get
  (And
    (Present
      (Inheritance (Variable "$X") (Variable "$Y1"))
      (Inheritance (Variable "$Y2") (Variable "$Z")))
    (Equal (Variable "$Y1") (Variable "$Y2"))))
```

That is joint variables are replaced by variable occurrences. Here `$Y` is replaced by variable occurrences `$Y1` and `$Y2`. This allows us to express the probability estimate as the product of the probabilities of the clauses times the probabilities of equality of the variables occurrences.

The formal definition of spec is left due to lack of place in the document.

This allows to see how such calculation can be entirely expressed as a reasoning process, although in practice it is wrapped into a single inference rule for efficiency purpose.

4.2 Beyond Independence-based Surprisingness and Dynamic Measure

Surprisingness has to be dynamic be nature, once a fact is known it is no longer surprising, thus...

The advantage of framing measuring surprisingness as a reasoning process is that we can easily make the transition from a hard coded definition of surprisingness to a more general one.

5 Experiment: Mining Surprising Patterns in SUMO

We have experiemented our current surprisingness measure on synthesized data (to verify the correctness of the algorithm described in REF), as well as real world data, in particular on the SUMO ontology REF. Most surprising patterns discovered are rather abstract and thus difficult to interpret, in spite of being surprising. Such patterns are for instance

TODO

Also patterns that are concrete enough to be understood by a human, such as

TODO (maritim)

are usually not surprising to a human because, although they are indeed distant from their probability estimate under independence assumptions, thus considered as surprising to the machine, are not surprising to us because we can use our background knowledge of the world as well as our capacity to infer a better estimate to be able to not be surprised.

6 Conclusion

It is clear from that experiment that the next step is to enable some form of open-ended (yet reasonably fast) means of inferences as to take into account more semantic knowledge, and thus .

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases (1994)
2. Chi, Y., Muntz, R., Nijssen, S., N. Kok, J.: Frequent subtree mining - an overview. Fundam. Inform. **66**, 161–198 (01 2005)
3. Derezinski, M., Rohanimanesh, K., Hydrie, A.: Discovering surprising documents with context-aware word representations. Proceedings of the 23rd International Conference on Intelligent User Interfaces, IUI 2018, Tokyo, Japan, March 07-11, 2018 pp. 31–35 (2018)
4. Endres, D., Schindelin, J.: A new metric for probability distributions. Information Theory, IEEE Transactions on **49**, 1858 – 1860 (08 2003)
5. Goertzel, B., Ikle, M., Goertzel, I.F., Heljakka, A.: Probabilistic Logic Networks. Springer US (2009)
6. Jeffreys, H.: An Invariant Form for the Prior Probability in Estimation Problems. Proceedings of the Royal Society of London Series A **186**, 453–461 (1946)
7. Pienta, R., Lin, Z., Kahng, M., Vreeken, J., Talukdar, P.P., Abello, J., Parameswaran, G., Chau, D.H.P.: Adaptivenav: Discovering locally interesting and surprising nodes in large graphs. IEEE VIS Conference (Poster) (2015)