Learning PAC Implication Basis using Formal Concept Analysis and Concept-Learning

Gaurav Sahu

January 31, 2018

1 Introduction

In this article, we will see how to combine features from the domain of Formal Concept Analysis and Concept-Learning to learn PAC implication basis. Before defining what an implication basis is, let us first recall what Formal Concept Analysis and Concept-Learning are:

1.1 Formal Concept Analysis (FCA)

Formal Concept Analysis is a branch derived from mathematics usually employed for analyzing data. Data (formally referred to as **context**) here, is represented as a **concept lattice** where each node of the lattice represents a different **concept**. A concept in itself is a relation between the *objects* and *attributes* in the data. A Formal definition of some important terms follows:

- **Context:** A formal context is a triple $\mathbb{K} = (G, M, I)$, where G and M are a set and $I \subseteq (G \times M)$ is a bipartite relation. The elements of G are called *objects* and those of M are called *attributes*. If an object $g \in G$ and an attribute $m \in M$ are in relation I, they are represented as $(g, m) \in I$ and the relation is read as *"the object g has the attribute m"*.
- **Concept:** (A, B) is a formal concept of $\mathbb{K}$ iff $A \subseteq G, B \subseteq M, A' = B, B' = A$. As we can see, a formal concept is made up of two parts: the first being the *extent* and the latter being the *intent*. Formally speaking, an extent is the set of objects *having* all the attributes of the intent which in turn, is the set of attributes *common* to all the objects in the extent. The derivation operator $(')$ used in the definition can produce two types of results: if applied to a set of objects, it would return its intent and if applied to a set of attributes, it would return its extent.
- **Closed:** The sets A and B are said to be *closed* in context $\mathbb{K}$ if $A = A''$ and $B = B''$, respectively. Such set of subsets of M closed in $\mathbb{K}$ is called the set of *intents* of $\mathbb{K}$ and is denoted by *Int(*$\mathbb{K}$*)*.

1.2 Concept-Learning

Concept-Learning is a technique which uses *queries* or *oracles* to learn an unknown concept from a given set of hypotheses. It is assumed that the learning algorithm has access to a fixed set of *oracles* that will answer some specific types of queries about the unknown concept we want to learn. The idea of using oracles for learning is not new, it dates back to 1988 when Angluin first brought it up[1]. There are many types of oracles but we will restrict our discussion to the following two:

- **Membership:** The input is an element $x \in U$ and the output is *true* if $x \in L_*$ and *false* if $x \notin L_*$, where U is the universal set of concepts and L_* is the unknown concept we wish to learn.
- **Equivalence:** The input is a hypothesis L and the output is *true* if $L = L_*$. If $L \neq L_*$, the oracle returns an element $x \in L \oplus L_*$. Here, x is called a *positive counterexample* if it belongs to L_* and a *negative counterexample* if it belongs L.

Now that we have recalled some important concepts required for further discussion, we shall discuss what PAC implication bases are and how to learn them using FCA and oracles.

2 What is a PAC Implication Basis?

In order to define PAC Implication Basis, we need to familiarize ourselves with some notions.

- **Implication:** An *implication* over M is a mapping $X \to Y$ where $X, Y \subseteq M$. The set of all implications over M is denoted by $Imp(M)$. In simple words, it can be interpreted as follows: if a set of objects has all the attributes of X, it will have all the attributes of Y also.
- **Closed:** A set $A \subseteq M$ is said to be *closed* under an implication $X \to Y$ if $X \nsubseteq A$ or $Y \subseteq A$. Such a set A is also called a *model* of the implication $X \to Y$ and the implication $X \to Y$ is said to *respect* the set A. Extending the definition, we say that a set A is closed under a set of implications $\mathcal{L}$ if it is closed under every implication $i \in \mathcal{L}$. The set of all such sets closed under $\mathcal{L}$ make up the model of $\mathcal{L}$, denoted by $Mod(\mathcal{L})$
- **Valid:** An implication $X \to Y$ is *valid* in $\mathbb{K}$ if attributes of all the objects (technically, $\{g\}'$) are closed under $X \to Y \ \forall \ g \in G$. Extending the definition, we say that a set of implications $\mathcal{L}$ is valid in $\mathbb{K}$ if all the implications in $\mathcal{L}$ are valid in $\mathbb{K}$.
- **Follows:** An implication $X \to Y$ *follows* from another implication $\mathcal{L}$ if for all contexts where $\mathcal{L}$ is valid, $X \to Y$ is also valid. It can be seen as an extension of *implication* from sets of attributes to the sets of implications.
- **Pseudo-Intent:** A set $P \subseteq M$ is called a *pseudo-intent* if $P \neq P''$ and each pseudo-intent $Q \subset P$ satisfies $Q'' \subseteq P$. We see that it doesn't follow the closed condition of the intents but still has the features of an attribute set and hence, the name *pseudo-intent*.
- **Basis and Canonical Basis:** An implication set $\mathcal{L} \subseteq Imp(M)$ is called a *basis* of the context $\mathbb{K}$ if $\mathcal{L}$ is *valid* in $\mathbb{K}$ and every implication valid in $\mathbb{K}$ *follows* from $\mathcal{L}$. A basis is called *minimal* if there exists no other implication set with lesser elements than $\mathcal{L}$. For a context $\mathbb{K} = (G, M, I)$ with finite G and M, the minimal basis is given by the *canonical-basis*, which is nothing but a set of all the pseudo-intents. Formally, a canonical basis is given by: $Can(\mathbb{K}) = \{P \to P'' \mid P \ is \ pseudo-intent \ of \ \mathbb{K}\}$

For our discussion, the Implication Basis is same as the Canonical Basis and we will use them interchangeably. We are interested in finding the implication bases because they provide a very convenient way to represent a formal context. However, they are not easy to compute and all the existing deterministic algorithms suffer from scalability issues. Moreover, almost all the datasets we deal with today have some level of *noise* and the existing deterministic algorithms are unable to handle it efficiently. That is why for all practical purposes it suffices to compute *approximate* bases. Formally speaking, we compute *probably approximately correct* implication bases (PAC bases) of the context $\mathbb{K}$.

3 Computing Probably Approximately Correct bases

Since we are aiming to compute an approximation of the exact implication bases, we need to have some tolerance regarding the approximated basis. We denote this tolerance by ε, where $0 < \varepsilon < 1$. In other words, ε shows how much the approximated basis can deviate from the exact implication basis. Theoretically, the "deviation" can be measured using the Horn-distance which is given by:

$$\text{dist}(\mathcal{H}, \mathbb{K}) = \frac{|Mod(\mathcal{H}) \triangle Int(\mathbb{K})|}{2^{|M|}} < \varepsilon$$

Here, $\mathcal{H}$ is said to be an approximately correct basis of the formal context $\mathbb{K} = (G, M, I)$ with accuracy $\varepsilon > 0$. In order to define a PAC-basis, we need an additional parameter $\delta > 0$, which represents the *confidence*. A PAC-basis can be defined as follows:

PAC-basis: Let M be a finite set, $\mathbb{K} = (G, M, I)$ be a formal context and $\Omega = (W, \Upsilon, Pr)$ be a probability space. A random variable $\mathcal{H} : \Omega \rightarrow \Re(Imp(M))$ is called a probably approximately correct basis (PAC basis) with accuracy $\varepsilon > 0$ and $\delta > 0$ if $Pr(dist(\mathcal{H}, \mathbb{K}) > \epsilon) < \delta$.

We shall now look at a very well known algorithm called $HORN1$, which employs the *membership* and *equivalence* oracles to find the exact implication basis. The pseudo-code of the $HORN1$ algorithm is as follows:

Algorithm 1 HORN1(M, member?, equivalent?)

```
$\mathcal{H} \leftarrow \phi$
while $C \leftarrow equivalent?(\mathcal{H})$ is a counterexample do
  if some $A \rightarrow B \in \mathcal{H}$ does not respect C then
    replace all implications $A \rightarrow B \in \mathcal{H}$ not respecting $C$ by $A \rightarrow B \cap C$
  else
    find first $A \rightarrow B \in \mathcal{H}$ such that $C \cap A \neq A$ and member?$(C \cap A)$ returns false
    if $A \rightarrow B$ exists then
      replace $A \rightarrow B$ by $C \cap A \rightarrow B \cup (A \setminus C)$
    else
      add $C \rightarrow M$ to $\mathcal{H}$
    end if
  end if
end while
return $\mathcal{H}$
```

The key to implementing the $HORN1$ algorithm is to simulate the oracles[2] properly. If we go by the theoretical definition, it won't be possible as they assume that the oracles have access to the target hypothesis but that is what we are trying to learn here in the first place. So, we need to use some properties innate to the desirable sets.

3.1 Simulating the membership oracle

Simulating the membership oracle is easy, we just have to check whether the input set is *closed* in context $\mathbb{K}$. Hence, the membership oracle will look like:

Algorithm 2 member?($\mathcal{L}$)

```
if $\mathcal{L} == \mathcal{L}''$ then
  return True
else
  return False
end if
```

3.2 Simulating the equivalence oracle

Simulating an equivalence oracle is a bit counter-intuitive. The core task of the equivalence oracle is to generate appropriate counterexamples for the $HORN1$ algorithm. As we know, counterexamples are of two types: *positive* and *negative*. Let us look at how the equivalence oracle would look like:

Algorithm 3 equivalent?($M, \mathcal{H}$)

```
for $2^{|M|}$ times do
  $X \subseteq M$
  if (member?($X$) and $X \notin Mod(\mathcal{H})$) or (not member?($X$) and $X \in Mod(\mathcal{H})$) then
    return $X$
  end if
end for
return true
```

Algorithm 3 needs some further explanation. As discussed earlier, the core task of *equivalent?* is to provide counterexamples to the $HORN1$ algorithm. In simplified terms, a counterexample can be described as a subset of the attribute set M satisfying some constraints. Now, what are these constraints? The randomly sampled $X \subseteq M$ needs to be either a positive or a negative counterexample, i.e either X is present in the target hypothesis (which is checked by the *membership oracle*) or it is a model of the current hypothesis $\mathcal{H}$ (Formally, it is stated as $\mathcal{H}$ *respects* X). And this is what Algorithm 3 precisely does. It samples a subset from the uniform distribution of the powerset of M and returns it if is a "valid" counterexample. If no such counterexample is found, it means that the $\mathcal{H}$ is indeed our target hypothesis.

By now, most of the readers might have guessed the problems with the *equivalent?* algorithm. Let us go through them one by one:

- **Scalability:** The number of iterations in one call to the equivalence oracle increases exponentially with the size of attribute set M. So, if $|M|$ is large, $2^{|M|}$ becomes even larger which would make the algorithm very slow.
- **Trivial Looping:** The number of iterations in the **for** loop is quite trivial. We don't need to iterate through the entire powerset every time and more intelligent looping is required.

In order to mitigate the aforementioned issues of the equivalence oracle, the notion of *approx-equivalent?* was introduced which is employed by the PAC-basis generating algorithm.

3.3 Simulating the approx-equivalence oracle

The oracle maintains an internal counter i, which is the number of equivalence queries already used and adjusts the number of iterations inside the oracle. Let us see how to simulate the oracle:

Algorithm 4 approx-equivalent?(M, ε, δ)

```
$i \leftarrow 0$ // number of equivalence queries
**return** **function**$(\mathcal{H})$ **begin**
    $i \leftarrow i + 1$
    **for** $l_i$ times **do**
      $X \subseteq M$
      **if** (member?$(X)$ **and** $X \notin Mod(\mathcal{H})$) **or** (**not** member?$(X)$ **and** $X \in Mod(\mathcal{H})$) **then**
            **return** $X$
      **end if**
    **end for**
    **return** **true**
```

Here, $l_i = \left\lceil \frac{(i - \log_2 \delta)}{\varepsilon} \right\rceil$ as proved in the joint work by Borchmann et.al. [3]. This approx-equivalence oracle is used by the PAC-basis generating algorithm as shown below:

Algorithm 5 pac-basis$(M, member?, \varepsilon, \delta)$

```
**return** HORN1$(M, member?, approx - equivalent?(member?, \varepsilon, \delta))$
```

The $pac - basis$ algorithm is implemented in terms of $HORN1$ and it allows computing PAC bases in time polynomial in $|M|$, output $\mathcal{L}$, $1/\varepsilon$ as well as $1/\delta$. We can see how *approx-equivalent?* helps optimize the generation of counterexamples which it turn makes $pac - basis$ much faster than the native $HORN1$ algorithm.

4 Conclusion

In this article, we familiarized ourselves with many notions in the literature of Formal Concept Analysis and Concept-Learning. We saw how to represent data as *concept-lattices* and how to simulate the various types of *oracles* . Finally, we integrated the features from the two fields to compute probably approximately correct implication basis (PAC basis) for a given context $\mathbb{K}$ which can be used to remove redundant definitions from the input data.

References

[1] Dana Angluin. "Queries and concept learning". In: *Machine learning* 2.4 (1988), pp. 319–342.

[2] Konstantin Bazhanov and Sergei Obiedkov. "Optimizations in computing the Duquenne–Guigues basis of implications". In: *Annals of mathematics and artificial intelligence* 70.1-2 (2014), pp. 5–24.

[3] Daniel Borchmann, Tom Hanika, and Sergei Obiedkov. "On the Usability of Probably Approximately Correct Implication Bases". In: *International Conference on Formal Concept Analysis*. Springer. 2017, pp. 72–88.