Learning from Instance-level Relationships

(Kevin) Yuk-Lap Yip

March 29, 2006

Abstract

A large proportion of research work on supervised learning has been focused on learning from positive and negative examples. In some situations, for example in some bioinformatics applications, there exists some information about the relationships between different objects (data instances) that can be used in learning, but has been under-utilized. In this project we mainly study two such kinds of instance-level relationships: must-links and cannot-links. A must-link specifies that two objects are both in the target concept or are both not in it, and a cannot-link specifies that exactly one of two objects is in the target concept. We assume that the relationships are provided by some active and passive oracles, the former accepts queries about the relationship between two objects and about the target concept, and the latter returns the relationships between random object pairs. We develop a generic algorithm for learning a target concept completely using linear number of oracle calls, time and space. Based on this generic method, we consider special situations in which we are allowed to use only active oracles, only passive oracles, and both. We also consider some practical issues, including the different cost and reward of different oracle calls, learning in a disjoint or non-disjoint multi-concept environment, and oracles that may be unable to answer some queries or may return incorrect relationships.

1 Introduction

This project studies the problem of learning a target concept from instance-level relationships in a theoretical setting. An instance-level relationship is a relationship between a set of data instances (objects). We will mainly study two types of instance-level relationships: must-links and cannot-links [18], both of which are binary relationships (relationships between two objects). A must-link between two objects specifies that the two objects are both in the target concept or are both not in the concept. A cannot-link between two objects specifies that exactly one of the two objects is in the target concept. The terms "must-link" and "cannot-link" originate from their more general definitions in a disjoint multi-concept scenario, in which the goal is to learn a set of disjoint concepts (classes) [1], and there is a must-link between two objects if and only if they are in the same concept. We can view the current problem as a disjoint two-concept scenario, in which the target concept is the first one, and its complement is the other.

There are practical situations in which the cost of obtaining a must-link or cannot-link is much lower than that of knowing whether an object is in the target concept. For example, given a set of genes, one important problem is to identify the ones that are essential, i.e., the ones of which the malfunctioning would be lethal to the organism. It could be quite costly to

[1] We avoid calling a concept a "class" in order not to get confused with the concept classes, which are sets of concepts.

find out whether a gene is essential since this may involve manual low-throughput laboratory experiments. On the other hand, if the translated protein sequences of two genes are very similar, or if their expression patterns are highly correlated, it might be reasonable to predict that they are both essential or are both non-essential, i.e., there is a must-link between them. Similarly, when predicting whether a protein has a certain function, if the expression patterns of two genes are completely uncorrelated, it is reasonable to assume that they have different functions, i.e., there is a cannot-link between them. Testing whether two sequences are similar can be done by running a sequence comparison program at low cost, while expression patterns can be obtained from high-throughput experiments that have much lower cost than doing a large number of low-throughput experiments. Therefore in such situations, it would be beneficial if object membership tests can be replaced by a reasonable number of instance-level relationship tests.

Now we give some formal definitions. Given a set X of objects, a concept class C, and a target concept $c \subseteq X$, there is a must-link between a pair of objects $x_1, x_2 \in X$ if and only if $(x_1 \in c \wedge x_2 \in c) \vee (x_1 \notin c \wedge x_2 \notin c)$. There is a cannot-link between a pair of objects $x_1, x_2 \in X$ if and only if $(x_1 \in c \wedge x_2 \notin c) \vee (x_1 \notin c \wedge x_2 \in c)$. As a shorthand, we write an instance-level relationship between two objects x_1 and x_2 as $((x_1, x_2), r)$, where $r = 1$ if it is a must-link, and $r = -1$ if it is a cannot-link. The notation (x_1, x_2) refers to an unordered pair of objects, such that (x_1, x_2) is identical to (x_2, x_1). We treat $((x_1, x_2), r)$ as a Boolean variable whose value depends on the target concept c. There are some obvious properties of instance-level relationships:

Property 1 Well-defined concepts:
For any object $x \in X$, $((x, x), 1)$ *is true.*

Property 2 Complementarity:
For any objects $x_1, x_2 \in X$ *and* $r \in \{-1, +1\}$, $((x_1, x_2), r) \Leftrightarrow \neg((x_1, x_2), -r)$.

That is, exactly one of the two types of relationships holds for each pair of objects.

The task is to learn c from the information supplied by some oracles. We consider several kinds of oracles:

- Active relationship oracle (AR): a function $AR : X \times X \rightarrow \{true, false\}$ that takes two objects x_1 and x_2 as inputs, and returns true if and only if (x_1, x_2) is a must-link. When there is a need to explicitly specify the underlying concept of an oracle, we add the name of the concept as a subscript. For example, AR means the same oracle as AR_c.

- Passive relationship oracle (PR): it takes no inputs, and returns a true relationship $((x_1, x_2), r)$, where (x_1, x_2) is a pair of distinct objects drawn randomly from a probability distribution over all pairs of distinct objects in X.

- Equivalence oracle with relationships as counter-examples (ER): it takes a representation of a concept c' as input, and returns true if c and c' are equivalent subject to permutation of the concepts, i.e., $c = c'$ or $c = X - c'$. Otherwise, ER returns false and a relationship $((x_1, x_2), r)$ such that $x_1 \neq x_2$ and $AR_c(x_1, x_2) \neq AR_{c'}(x_1, x_2)$. The relationship is called a counter-example. The pair (x_1, x_2) is drawn randomly from a probability distribution over all pairs of distinct objects in X.

To compare with some previous results, we also consider the corresponding oracles based on memberships, i.e., whether an object is in the target concept c:

- Active membership oracle (AM): a function $AM : X \rightarrow \{true, false\}$ that takes an object x as input, and returns true if and only if $x \in c$.
- Passive relationship oracle (PM): it takes no inputs, and returns a pair (x, m), where x is an object drawn randomly from a probability distribution over X, and $m = 1$ if $x \in c$, $m = -1$ if $x \notin c$.
- Equivalence oracle with memberships as counter-examples (EM): it takes a representation of a concept c' as input, and returns true if $c = c'$. Otherwise, EM returns false and a true membership (x, m), where $x \in X$ and $m = 1$ if $x \in c$, $m = -1$ if $x \notin c$, such that $AM_c(x) \neq AM_{c'}(x)$. The membership is called a counter-example. The object x is drawn randomly from a probability distribution over X.

Efficient algorithms exist [2, 3] that learn concepts of some concept classes from AM and EM only. A major goal of this project is to study the feasibility of learning by other combinations of the oracles. This involves understanding the relationships between the three relationship oracles and their membership counterparts. The passive oracles are to simulate situations in which some knowledge about the target concept is available and can be acquired at some cost, but there are no simple ways to answer specific questions about the concept.

2 Related work

Gold's classical paper on language identification [12] studies the identification of a language in the limit by using various kinds of text and informant, where a text is a sequence of positive examples such that each object in the target concept appears at least once, and an informant is a sequence of examples (positive or negative), with some requirements on the sequence according to the informant type. Both text and informant are membership information about individual objects. In Valiant's paper that describes the probably approximately correct (PAC) learning model [17], concepts are learnt from positive examples and active membership oracles, which are also membership information about individual objects.

Angluin's papers on learning from queries [2, 3] brought up the idea of using some other kinds of information in machine learning, namely the relationships between the target concept and the current hypothesis, such as equivalence and disjointness. We will call such information "language-level information".

Since then, however, a large proportion of research efforts in machine learning have been focused on learning from positive and negative examples only. This problem has been so popular that the term "supervised-learning" is now almost equivalent to "classification", or "learning from positive and negative examples"[2], despite the fact that a learner can be "supervised" by other means. The many well-known machine learning methods, such as decision trees, k-nearest neighbors, neural networks, naïve Bayes classifiers, and support vector machines, are all designed for this setting.

[2] In a multi-concept scenario, this is generalized as "learning from labeled examples".

On the other extreme, many researchers have focused on unsupervised learning, which is to learn some concepts without receiving any information other than the dataset itself, which contains a set of objects, each described by a set of attributes. There is a controversy about whether this is a real learning problem, since most, if not all, natural learning processes involve some kinds of feedback, from a dedicated teacher or from the environment. As a result, the problem is now more commonly known as clustering, and is vaguely defined as a process to group the data objects such that objects in the same cluster are similar, and objects in different clusters are dissimilar.

Recently, the distinction between supervised and unsupervised learning methods has become less obvious due to the invention of semi-supervised algorithms. They arose due to practical situations in which both classification and clustering methods are not ideal for learning. On the one hand, classification is inappropriate when the information available is not in the form of positive and negative examples, or the examples are too few or too biased to capture the general properties of the target concept. On the other hand, clustering completely ignores all external knowledge, which could make it inappropriate when some external knowledge is necessary for producing the desired clusters. According to the design objectives, semi-supervised methods can be classified into semi-supervised classification and semi-supervised clustering methods.

Most current work on semi-supervised classification focuses on exploiting the data attributes of unlabeled objects (objects not known to be in the target concept or not) in improving the classification performance. In the co-training setting [8], each data object has two sets of data attributes corresponding to two kinds of information. For example, each web page can be described by two sets of attributes, one corresponding to the text on the web page, the other corresponding to the anchor text attached to the hyperlinks that point to the web page. If class labels can be assigned based on one set of attributes alone, then assigning a class label to an object based on its values in the first set of attributes implies that other objects having the same attribute values in the second set as this object can also be given the same label. This technique has been used in a practical way as follows: label some objects according to the first set of attributes. Search for objects that have values in the second set of attributes very close to the labeled objects, and give them the same labels. Then reverse the roles of the two sets, and repeat.

Another popular semi-supervised classification method is based on a Markov random walk [15]. The dataset is modeled as a graph, with each vertex being an object of the dataset. There is an edge between two objects if the similarity between them exceeds a certain threshold. Starting with only a few examples with known labels, the labels diffuse probabilistically to neighboring vertices according to the similarity between the current object and the neighbor. After a certain number of time steps, each vertex is associated with a certain probability of being a member of each concept. The object is then predicted to be a member of the concept with the highest probability. This method can be viewed as a generalization of k-nearest neighbors, but with a more extensive use of object similarities.

In bioinformatics, there is a popular program called PSI-BLAST [1] for searching for sequences that share a common motif. First, some example sequences are used to search for similar sequences. Then the sequences in the results are added to the examples for the next round of similarity search. The process repeats until no new sequences are returned. The methodology is similar to that of the semi-supervised classification methods, although it was not developed based on the corresponding computational learning theories.

Semi-supervised clustering methods aim at utilizing some domain knowledge in guiding the

clustering process. They can be categorized according to the kinds of knowledge being input, the time that the knowledge is input, and the way the knowledge is used to affect the clustering process.

The simplest type of inputs is labeled objects [4, 11, 21], which is the same as the examples used in classification. In some cases, users do not know the exact class labels of objects, but they have some knowledge on which objects should be/should not be put into the same cluster, which can be specified by must-links and cannot-links [5, 6, 7, 13, 18, 19]. In projected clustering, the input of known relevant dimensions is also useful [21]. Some other studies propose the input of classification rules [16], examples of similar objects [20], or even general comments such as which cluster a particular object should not be put into [10].

The knowledge can be supplied at different time. It can be supplied before clustering to guide the clustering process [4, 5, 6, 7, 11, 13, 18, 19, 20, 21], or after clustering to evaluate the clusters and guide the next round of clustering [10]. Some algorithms can also actively request users to supply some specific information at the most appropriate time [5, 13].

Some algorithms treat the input knowledge as hard constraints that should not be violated [4, 19], but most treat it as soft constraints, as it may contain errors. There are various ways to use the input knowledge, such as guiding the formation of seed clusters [4, 5, 6, 7, 21], forcing or recommending some objects to be put into the same cluster or different clusters [18, 19], and modifying the objective function [5, 6, 7, 11], similarity function [6, 7, 10, 20] or distance matrix [13].

In general, a new trend in machine learning algorithm design is to maximize the use of all available information: labeled examples, data attributes, language-level information, instance-level relationships, and so on. While instance-level relationships have been used in practical semi-supervised clustering algorithms, as far as the author is aware of, they have not been studied from a theoretical perspective. This project is an initial attempt to study the properties of the two types of instance-level relationships, must-links and cannot-links, in a theoretical setting. It is hoped that the current project would also shed some light on the larger goal of understanding semi-supervised learning methods.

3 A general learning method

We now study a basic problem of learning from instance-level relationships. The target concept c can be represented as a graph $G = (V, E)$, where $V = X$ and $E = \{(x_1, x_2) : x_1, x_2 \in X \wedge AR_c(x_1, x_2)\}$, i.e., the set of all must-links (again, if we need to specify the concept explicitly, we use subscripts, e.g., $G_c = (V_c, E_c) = G = (V, E)$). We call G the (unique) relationship graph of c. G contains two cliques[3], one for the objects in c and one for the objects not in c. The complement of G, $\tilde{G} = (V, \tilde{E}) = (V, V \times V - E)$ is a bipartite graph, with every vertex in the first part connecting to every vertex in the second part. Each edge represents a cannot-link. We will name the vertices by the objects that they represent. For example, the vertex that represents the object x_1 will also be called x_1.

If we know the exact members in c, we know the exact members in E. Therefore identifying E is a necessary condition for identifying c. The basic learning problem that we are going to study is as follows: at time 0, we start with two empty sets $E^0 = \tilde{E}^0 = \emptyset$, and make an oracle call (AR, ER or PR) to get some information. Then we update the two sets by adding all

[3]In case $c = X$ or $c = \emptyset$, one of the cliques has size zero. The complement $\tilde{G}$ then has one part empty.

newly known must-links to E^0 to form E^1 and all newly known cannot-links to $\tilde{E}^0$ to form $\tilde{E}^1$. The process is repeated until a time t^* at which $E^{t^*+1} = E$ and $\tilde{E}^{t^*+1} = \tilde{E}$. A brute force approach to learning E is to make an AR call for every pair of objects. Since there are $O(|V|^2)$ object pairs, this brute force approach requires $O(|V|^2)$ oracle calls.

The worst-case number of oracle calls required is actually much smaller than $O(|V|^2)$. This is because existing knowledge can derive new knowledge without making oracle calls. All these derivations are based on the following three properties:

Property 3 Transitivity of must-links:
For any objects $x_1, x_2, x_3 \in X, ((x_1, x_2), 1) \wedge ((x_2, x_3), 1) \Rightarrow ((x_1, x_3), 1)$.

Property 4 Propagation of cannot-links:
For any objects $x_1, x_2, x_3 \in X, ((x_1, x_2), 1) \wedge ((x_2, x_3), -1) \Rightarrow ((x_1, x_3), -1)$.

Property 5 Elimination of cannot-links:
For any objects $x_1, x_2, x_3 \in X, ((x_1, x_2), -1) \wedge ((x_2, x_3), -1) \Rightarrow ((x_1, x_3), 1)$.

Therefore a single oracle call may result in multiple newly known must-links and/or cannot-links. Notice that each rule is able to infer some knowledge that cannot be inferred by the other two rules.

There is an interesting way to summarize the three rules into a single statement: every three objects in X form an odd number of must-links and an even number of cannot-links. Putting it into an equation, suppose the relationships of the three pairs are $r_{1,2}$, $r_{1,3}$ and $r_{2,3}$ respectively, then $r_{1,2}r_{1,3}r_{2,3} = 1$. It can be easily verified that the above three rules are the only non-trivial inference rules that can be derived from this statement. A nice thing about the three properties is that for every three objects, whenever we have the relationships between two of the three pairs, we can always deduce the relationship between the third pair.

We are going to describe learning algorithms that use different combinations of oracle calls in the coming sections. Suppose now that we have no control of what oracle calls to make. Then the only thing that we can do is to infer all relationships that can be inferred after each oracle call. This can be achieved as follows. Suppose the oracle call at time t is $ER(c')$ and the oracle returns true, then E and $\tilde{E}$ can be immediately deduced and we are done. If the oracle returns false and in cases where the call is made to AR or PR, we get the relationship between a pair of objects x_1 and x_2. If the relationship is not already known, we add it to E^t or $\tilde{E}^t$ according to its type, and use the three rules to infer an edge for each existing edge that involves x_1 or x_2. We can easily show that one single pass of inference is sufficient to infer all new edges that can be inferred. Suppose there is an existing edge (x_1, x_3) so that together with the new pair (x_1, x_2) the relationship between x_2 and x_3 can be inferred. If there is an edge between x_3 and an object x_4, then we can further infer the relationship between x_2 and x_4. However, since the relationships between x_1 and x_3 and between x_3 and x_4 are both known before the current oracle call, the relationship between x_1 and x_4 should have already inferred. Therefore the relationship being "further" inferred can actually be directly inferred from the two pairs (x_1, x_4) and (x_1, x_2), which is covered by the first pass of inference.

Suppose in each oracle call we receive a relationship that is not already known. We want to analyze the worst-case number of oracle calls required. But in order to make the analysis simple, let us first discuss a practical implementation of the above procedure. A direct implementation requires $\Theta(|V|^2)$ time and space, because of the quadratic number of edges we need to create

in E^t and $\tilde{E}^t$. There is a simple alternative that runs in linear time and space as follows. Whenever we are to add an edge (x_1, x_2) to E^t, instead we merge x_1 and x_2 into a single vertex $x_{1,2}$ in both G^t and $\tilde{G}^t$. All edges originally incident on x_1 or x_2 in $\tilde{G}^t$ are then incident on $x_{1,2}$. Suppose after performing all possible merges at time t, the resulting set of vertices is V^{t+1}. Due to Property 3, for any vertex x_3 and any time $t' > t$, the vertex that represents both x_1 and x_2 also represents x_3 in this new implementation if and only if $(x_1, x_3) \in E^{t'}$ and $(x_2, x_3) \in E^{t'}$ in the direct implementation. Similarly, due to Property 4, for any vertex x_3 and any time $t' > t$, $(x_{1,2}, x_3) \in \tilde{E}^{t'}$ in the new implementation if and only if $(x_1, x_3) \in \tilde{E}^{t'}$ and $(x_2, x_3) \in \tilde{E}^{t'}$ in the direct implementation. Therefore the new implementation changes the representation of the graphs, but does not alter the procedure.

We now analyze the time and space complexities of this new implementation, as well as the number of oracle calls required to completely determine E. For simplicity, if a vertex represents multiple objects, we can call it by any of the objects. For example, after merging the vertices x_1 and x_2, the three symbols $x_{1,2}$, x_1 and x_2 are all valid names of the resulting vertex. Suppose the information received at time t is $((x_1, x_2), r)$, then there are only five different cases of graph updates:

1. $r = 1$, and either x_1 or x_2 does not have any incident edge in $\tilde{G}^t$. In this case, we simply merge x_1 and x_2. Therefore $|V^{t+1}| = |V^t| - 1$, and $|\tilde{E}^{t+1}| = |\tilde{E}^t|$.

2. $r = 1$, and both x_1 and x_2 have an incident edge in $\tilde{G}^t$. Suppose they are (x_1, x_3) and (x_2, x_4) respectively, then (x_3, x_4) is inferred to be a must-link. In this case, we merge x_1 and x_2, and x_3 and x_4. The two cannot links (x_1, x_3) and (x_2, x_4) then become the same edge in $\tilde{G}^{t+1}$. Therefore $|V^{t+1}| = |V^t| - 2$, and $|\tilde{E}^{t+1}| = |\tilde{E}^t| - 1$.

3. $r = 0$, and both x_1 and x_2 do not have any incident edge in $\tilde{G}^t$. In this case, we simply add an edge between x_1 and x_2 to $\tilde{G}^{t+1}$. Therefore $|V^{t+1}| = |V^t|$, and $|\tilde{E}^{t+1}| = |\tilde{E}^t| + 1$.

4. $r = 0$, and exactly one of x_1 and x_2 have an incident edge in $\tilde{G}^t$. Suppose it is (x_1, x_3), then (x_2, x_3) is inferred to be a must-link. In this case, we merge x_2 and x_3, and the new cannot-link edge between x_1 and x_2 becomes the same as the existing edge between x_1 and x_3. Therefore $|V^{t+1}| = |V^t| - 1$, and $|\tilde{E}^{t+1}| = |\tilde{E}^t|$.

5. $r = 0$, and both x_1 and x_2 have an incident edge in $\tilde{G}^t$. Suppose they are (x_1, x_3) and (x_2, x_4) respectively. Then (x_2, x_3) and (x_1, x_4) are inferred to be must-links. In this case, we merge x_1 and x_4, and x_2 and x_3, and the new cannot-link edge between x_1 and x_2 becomes the same as the two existing edges between x_1 and x_3 and between x_2 and x_4. Therefore $|V^{t+1}| = |V^t| - 2$, and $|\tilde{E}^{t+1}| = |\tilde{E}^t| - 1$.

If each vertex in $\tilde{E}^t$ has at most one edge incident on it, it can be easily verified that the above list includes all possible graph updates that can be performed at time t. We now prove that this sufficient condition is true:

Lemma 1 *At any time t, each vertex in $\tilde{G}^t$ can have at most one edge incident on it.*

Proof. We prove it by contradiction. Suppose t is the first time that there is a vertex in $\tilde{G}^t$ with two different edges incident on it. Then a must-link is inferred between the two neighbors, and thus the updating procedure guarantees that they should have been merged at a time $t' < t$

because at that time each vertex in $\tilde{G}^{t'}$ has at most one edge incident on it and thus the above list of graph updates is exhaustive at that time, which is a contradiction. Therefore such t does not exist and thus the lemma follows. □

Based on the above results, after completing the merges at a given time, all objects known to be must-linked at that time are put into the same vertices.

Corollary 1 *At any time t, $|\tilde{E}^t| \leq \lfloor \frac{|V^t|}{2} \rfloor$.*

The following lemma concerns the termination condition.

Lemma 2 *When the learning algorithm terminates, if $c = X$ or $c = \emptyset$, then $|V^{t^*+1}| = 1$ and $|\tilde{E}^{t^*+1}| = 0$. Otherwise, $|V^{t^*+1}| = 2$ and $|\tilde{E}^{t^*+1}| = 1$.*

Proof. The algorithm terminates when $E^{t^*+1} = E$. The lemma then follows form the facts that 1) G contains one single clique if $c = X$ or $c = \emptyset$, and two cliques otherwise; and 2) all known must-linked objects are put into the same vertices after completing the merges at any given time. □

Finally, the following lemma concerns the total number of received and inferred must-links.

Lemma 3 *The total number of distinct must-links between different vertices received from oracles and inferred by the three inference rules is exactly $|V| - 1$ if $c = X$ or $c = \emptyset$, and exactly $|V| - 2$ otherwise.*

Proof. The lemma is based on Lemma 2, and the facts that 1) $|V^0| = |V|$; 2) $|V|$ is reduced only when a must-link is received or inferred; and 3) each received or inferred must-link reduces $|V|$ by exactly one. □

If we denote the number of times that the five cases occur as m_1, m_2, m_3, m_4 and m_5 respectively, then from Lemmas 2 and 3, by considering the change of $|V^t|$ we know that $m_1 + 2m_2 + m_4 + 2m_5 = |V| - |\tilde{E}^{t^*+1}|$, and from Lemma 2, by considering the change of $|E^t|$ we know that $-m_2 + m_3 - m_5 = |\tilde{E}^{t^*+1}|$. Summing the two equations, we get the total number of oracle calls, $m_1 + m_2 + m_3 + m_4 + m_5 = |V| - |\tilde{E}^{t^*+1}| + |\tilde{E}^{t^*+1}| = |V| - 1$.

We summarize the result in the following theorem.

Theorem 1 *For any concept c defined over a set of objects X, the corresponding relationship graph $G = (X, \{(x_1, x_2) : x_1, x_2 \in X \wedge AR_c(x_1, x_2)\})$ can be learnt from any sequence of $|X| - 1$ calls to AR, ER and PR with arbitrary arguments using $O(|X|)$ time and $O(|X|)$ space if every call returns some new information.*

Proof. We have just proved the number of oracle calls. Then, since each update involves a constant number of operations and uses a constant amount of extra space, the time and space complexities follow. Corollary 1 gives a tight bound on the space required, which happens when the first $\lfloor \frac{|X|}{2} \rfloor$ oracle calls all lead to the third case of graph updates. □

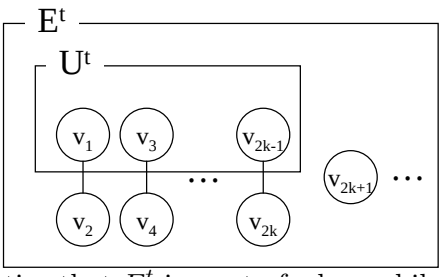

Figure 1: Definition of U^t. Notice that E^t is a set of edges while U^t is a set of objects.

During the development we assumed that we know exactly when $E^{t^*+1} = E$ and $\tilde{E}^{t^*+1} = \tilde{E}$. Again, a straightforward implementation of the algorithm would perform an ER call after each update, which either increases the number of oracle calls required, or restrict the sequence of oracle calls. This check is actually not necessary because the termination condition can occur at time t only if $|V^t|$ equals one or two. Therefore we can proceed without checking the termination condition until $|V^t| = 2$, when we simply make one final oracle call, which ensures that at time $t + 1$, the termination condition must be satisfied.

Now, we study a strategy for making oracle calls that ensures we get some new information after each call. At any time t, we can freely choose to make either of the following calls:

- $AR(x_1, x_2)$, where x_1 and x_2 are represented by two different vertices in V^t, and $(x_1, x_2) \notin \tilde{E}^t$.
- $ER(U^t)$, where U^t is a set of objects formed by the following method. From Lemma 1, we know that no three vertices in $\tilde{E}^t$ have two edges in between. Therefore we can imagine that the vertices in $\tilde{E}^t$ are sorted such that the first vertex v_1 connects only to the second vertex v_2, the third only to the fourth, and so on. Let k be the number of such vertex pairs. Then $U^t = \cup_{i=1}^{k} v_{2i-1}$, where we overload the symbol v_i to mean the set of objects represented by vertex v_i (Figure 1).

In the first case, we do not know whether (x_1, x_2) is a must-link (otherwise they would be in the same vertex), or a cannot-link (otherwise (x_1, x_2) would be in $\tilde{E}^t$). Therefore the response of AR must contain some new information. In the second case, suppose ER returns false and the counter example is a must-link, the two objects involved must be represented by different vertices in V^t since all known must-linked objects are either both in U or both not in U. Therefore it is a new piece of knowledge. If the counter example is a cannot-link, the two objects involved must be both in U or both not in U, which is also a new piece of knowledge since all cannot-links known at time t are between an object in U and an object not in U.

We therefore can state the following theorem.

Theorem 2 *For any concept c defined over a set of objects X, the corresponding relationship graph $G = (|X|, \{(x_1, x_2) : x_1, x_2 \in X \wedge AR_c(x_1, x_2)\})$ can be learnt from any sequence of $|X|-1$ calls to AR and ER with arguments determined by the described strategy using $O(|X|)$ time and $O(|X|)$ space.*

One important feature of this algorithm is that no matter what oracle calls have been made previously, as long as E and $\tilde{E}$ has not been learnt completely, the algorithm can always

make an oracle call that guarantees to return some new information. Therefore, the algorithm can maximize the use of prior knowledge. Given any known instance-level relationships, the algorithm simply initializes V^0, E^0 and $\tilde{E}^0$ accordingly, and starts running as if the knowledge was obtained through oracle calls. Similarly, if some knowledge becomes freely available at any time during the execution, it can be imported to the algorithm as if it was obtained through oracle calls, so that after the import the algorithm can resume at a state that contains the union of knowledge from both the learning process and the import.

There are two pending issues to be discussed. First, although learning E is a necessary condition for learning c, it is not sufficient. Second, it is important to know the tightness of the upper bounds in Theorem 1 and Theorem 2. In other words, whether it is possible to learn E using fewer than $|X| - 1$ oracle calls in the worst case. Both topics will be discussed in the coming section.

4 Active Learning

In this section we study the problem of learning c actively, i.e., using AR and ER calls. We first discuss some relationships between AR and AM.

Property 6 Simulating AR by AM:
The call $AR(x_1, x_2)$ can be simulated by the calls $AM(x_1)$ and $AM(x_2)$ as follows: $AR(x_1, x_2) = (AM(x_1) \wedge AM(x_2)) \vee (\neg AM(x_1) \wedge \neg AM(x_2))$.

Property 7 Simulating AM by AR:
If there is a (positive or negative) example x of c, the call $AM(x')$ can be simulated by the call $AR(x, x')$ as follows: $AM(x') = (AR(x, x') \wedge (x \in c)) \vee (\neg AR(x, x') \wedge (x \notin c))$.

Now we switch to the equivalence oracles.

Property 8 Relationship between ER and EM:
If ER accepts a concept c' (i.e., returns true when taking c' as input), we can get a concept c'' that EM accepts if there exists one positive or negative example of c. If such an example is not available, it can be obtained by calling AM once using any object in X as input, or calling PM once.

This property follows from the fact that the relationship graph G contains two disconnected cliques, so by knowing the label of only one single object, the labels of all objects can be inferred. The property suggests that if there is an efficient algorithm to learn E, learning c afterwards is straightforward.

The above property also suggests ways for ER and EM to simulate each other.

Property 9 Simulating ER by EM:
The call $ER(c')$ can be simulated by the calls $EM(c')$ and $EM(X - c')$ as follows. The simulator returns true if either $EM(c')$ or $EM(X - c')$ returns true. Otherwise, it returns false. Suppose the counter examples from $EM(c')$ and $EM(X - c')$ are (x_1, m_1) and (x_2, m_2) respectively, then the counterexample for $ER(c')$ is $((x_1, x_2), m_1 m_2)$.

Notice that x_1 and x_2 must be different objects, since either one of them is in c and the other in $X - c$, or one of them is in c' and the other in $X - c'$.

Property 10 Simulating EM by ER and AR:
If there is a (positive or negative) example x of c, the call $EM(c')$ can be simulated by the call $ER(c')$ as follows. The simulator returns true if $ER(c') \wedge ((x \in c \wedge x \in c') \vee (x \notin c \wedge x \notin c'))$ is true. Otherwise, it returns false. Suppose the counter example from $ER(c')$ is $((x_1, x_2), r)$, we need to determine whether x_1 is in c by simulating $AM(x_1)$ by $AR(x, x_1)$ as discussed. Then the counter example for $EM(c')$ is based on the following table:

$x_1 \in c'$	$x_2 \in c'$	$x_1 \in c$	*Counter example*
True	*True*	*True*	$(x_2, -1)$
True	*True*	*False*	$(x_1, -1)$
True	*False*	*True*	$(x_2, 1)$
True	*False*	*False*	$(x_1, -1)$
False	*True*	*True*	$(x_1, 1)$
False	*True*	*False*	$(x_2, -1)$
False	*False*	*True*	$(x_1, 1)$
False	*False*	*False*	$(x_2, 1)$

Based on Property 7 and Property 10, if a positive or negative example is available at the beginning, any algorithm that can learn c using w_1 calls to AM and w_2 calls to EM regardless of the distribution of counter examples returned by EM can be simulated by using $w_1 + w_2$ calls to AR and w_2 calls to ER.

If the example is not available at the beginning, the simulation can be performed as follows. Pick one object $x' \in X$ as the reference point, and assume it is in c. Simulate each AM call by an AR call as described in Property 7, but EM calls are simulated by ER calls in a slightly different way. For each $EM(c')$ call, if $ER(c')$ returns false, returns false with the counter example discussed in Property 10. If $ER(c')$ returns true, we pause the simulation and wait for the example to come (or get one by an AM or PM call), and use the example to verify our assumption that x' is in c. If the assumption is correct, we simply returns true and c' is the correct hypothesis. Otherwise, c' must be equal to $X - c$, so we terminate the simulation, and report that $X - c'$ is the correct hypothesis.

We summarize the above results in the following theorem.

Theorem 3 *For any concept class C, if every concept can be learnt by making w_1 calls to AM and w_2 calls to EM regardless of the distribution of counter examples returned by EM, then every concept can be learnt by making $w_1 + w_2$ calls to AR, w_2 calls to ER, and at most one call to AM or PM.*

So, for example, since regular sets can be learnt from AM and EM in time polynomial in the number of states of the corresponding deterministic finite-state acceptor of the target concept and the maximum length of the counter examples without making any assumption of the distribution of counter examples returned by EM [2], they can also be learnt from AR and ER in polynomial time if we have one single string that is known to be in the target regular set or not.

Intuitively, since each response of an AM call and each counter example returned by an EM call contains the information of one single object only, while those for AR and ER contain the information about two objects, it seems reasonable to postulate that there are concept classes in which the concepts can be learnt by using fewer AR and ER calls then AM and EM calls.

The following example shows that such concept classes do exist if the argument to each EM call must be a valid concept in C.

Example 1 *Let $X = \{x_1, x_2, ..., x_n\}$ and C be the set of all $(n-1)$-subsets of X. For an algorithm that uses AM and EM only, in the worst case $n-1$ calls are needed to learn a concept. The worst case happens when all AM calls return true and all $EM(c')$ calls return false, with $X - c'$ as the positive counter example. On the other hand, if at any time the answer to an AR call is false, then only two concepts are still possible. So the worst case happens when all the answers to AR calls are true. In that case, each AR call eliminates two candidate concepts, so in total only $\lceil \frac{n}{2} \rceil$ AR calls are needed to learn any target concept in C.*

We can visualize the difference between the two approaches by a "learning tree", which is similar to a binary decision tree, but the set of training examples in each node is replaced by the set of consistent concepts, each attribute test is replaced by an oracle call, and each leaf node contains exactly one concept. If only AM and EM calls are allowed, each time one of the branches is a leaf node with a single concept. An adversary can thus manipulate the path of going down the tree such that it takes $n-1$ steps to reach a leaf node. On the other hand, each AR call has one of the branches being a tree with two leaf nodes. Thus the longest path that an adversary can produce has length $\lceil \frac{n}{2} \rceil$.

This "learning tree" analysis is similar to Littlestone's mistake tree analysis [14] in that in both cases:

- The learning process is to traverse along a path down a tree from the root to the leaf node that represents the target concept.
- We can imagine that the worst case scenario is due to an adversary who chooses the path of going down the tree (by choosing a concept) such that the total cost of the learner is maximized.
- An optimal learning algorithm is one that minimizes the worst-case total cost.

The differences between the two analyses are as follows:

- In the mistake tree case, three things occur at each tree node: the adversary chooses a test, the learner responds, and the adversary discloses the correct answer; while in the learning tree case, only two things occur: the learner posts a question and the adversary answers.
- In the mistake tree case, the total cost is the number of incorrect responses of the learner; while in the learning tree case, the total cost is the length of the traversed path.

The last point leads to an interesting phenomenon: the kind of unbalanced tree that an AM and EM algorithm would encounter in Example 1 is welcome by the learning tree adversary since it maximizes the maximum path length, but is hated by the mistake tree adversary since it minimizes the minimum number of mistakes, namely one. On the other hand, a complete tree is welcome by the mistake tree adversary since it maximizes the minimum number of mistakes, while it is hated by the learning tree adversary since it minimizes the maximum path length.

Since each AR call can be simulated by two AM calls, this example illustrates a situation in which the advantage of using relationship oracles over membership oracles is maximum.

From a theoretical point of view, if we ignore the constant factors, and assume that 1) we have one positive or negative example of the target concept some time before or after the learning process; and 2) the learning algorithm does not rely on the distribution of counter examples returned by equivalence oracles, then any AM and EM algorithm has a corresponding AR and ER algorithm that uses the same order of the number of oracle calls, and vice versa.

We now return to the question raised at the end of the last section: whether the upper bounds in Theorem 1 and Theorem 2 is tight. We first observe the fact that there are concept classes of which the concepts are intrinsically hard to learn:

Lemma 4 *If $C = 2^X$, i.e., C contains all possible subsets of X, then all oracle algorithms require at least $|X|$ oracle calls to learn the target concept c in the worst case, if there is no prior knowledge about c.*

Proof. The lemma follows from the fact that each oracle call can at most reduce the version space by one half in the worst case. □

Since after learning E, c can be learnt by making one additional oracle call, for the concept class $C = 2^X$, E cannot be learnt from fewer than $|X| - 1$ oracle calls. Therefore the upper bounds in Theorem 1 and Theorem 2 cannot be improved in general.

5 Relationship concept classes

In Section 3, we studied an algorithm for learning E from AR and ER calls. From the results in Section 4, we know that this algorithm can be simulated by AM and EM calls. What if we wanted to learn E from AM and EM directly in the first place? We could have formulated the problem as follows. Define the function f that takes a concept $c \in C$ as input and returns the set $E_c = \{(x_1, x_2) : x_1, x_2 \in X \wedge AR_c(x_1, x_2)\}$ of all must-links in c. Then we denote the range of f as $C_2 = \{f(c) : c \in C\}$. We will call C_2 the relationship concept class of C, and it is defined over (X, X) [4]. For a given concept $c \in C$, learning E_c is equivalent to learning $f(c)$. The relationship oracles for c become exactly the membership oracles for $f(c)$.

In bioinformatics, network (i.e., graph) learning has become a popular topic. Instead of learning some features of individual genes or proteins, more and more studies have switched to studying the relationships between different biological objects. For example, protein-protein interaction networks concern the interactions between different proteins, regulatory networks concern how transcription factors regulate the expression of genes, metabolic networks concern the interactions between enzymes and metabolites in metabolic pathways, co-expression networks concern the co-expression of genes, and co-essentiality networks concern genes that are co-essential. There is an edge between two objects if they have a certain kind of relationship, and the relationship can be due to some underlying features of the two individual objects. Putting in our current language, the relationships may be due to some hidden concepts, but what we can observe are the relationships rather than the concept memberships. Therefore it has become popular to treat each object pair as a higher order object, each relationship a higher order membership, and the set of all relationships the target concept to be learnt. The

[4] We write (X, X) instead of X^2 to emphasize that the order is unimportant, and that the size of the set is $\frac{|X|(|X|+1)}{2}$ instead of $|X|^2$.

class of such concepts is precisely the relationship concept class of the original concept class defined over the set of individual objects.

We begin the study by noticing that properties of must-links and cannot-links are automatically translated into properties of the relationship concept class C_2. For example,

Property 11 Concept size:
For any concept class C defined over the set X of objects and any concept $c \in C$, $|f(c)| = \frac{1}{2}|c|(|c|+1)+\frac{1}{2}|X-c|(|X-c|+1)$. Therefore each concept in C_2 has a size between $\lceil \frac{1}{4}|X|(|X|+1)\rceil$ and $\frac{1}{2}|X|(|X|+1)$.

Since C_2 is defined over (X, X), there are $2^{O(|X|^2)}$ possible concepts. Among them, only a small proportion satisfies the size requirement. And in fact, since C_2 is the image of C under f, the size of C_2 is at most $2^{O(|X|)}$. The huge difference between $2^{O(|X|^2)}$ and $2^{O(|X|)}$ shows that the properties of must-links and cannot-links impose a very strong constraint on the concepts in C_2.

More precisely, the maximum size of C_2 is $2^{|X|-1}$, since for all $c \in C, f(c) = f(X - c)$. Therefore, the VC-dimension of any relationship concept class C_2 is at most $|X| - 1$ (we simply do not have enough concepts to sustain a higher VC-dimension). We can show that it is a tight bound. Suppose $X = \{x_1, x_2, ..., x_{|X|}\}$ using some arbitrary ordering of the objects in X, then when $|C| = 2^{|X|}$, C_2 can shatter the following $(|X| - 1)$-subset of (X, X): $\{(x_1, x_2), (x_1, x_3), ..., (x_1, x_{|X|})\}$. This is because the set contains no two edges between any three objects in X, which means each pair in the set is independent of all others, i.e., it can freely choose its relationship regardless of the other pairs.

Therefore, if we treat the members in (X, X) as Boolean variables whose values depend on the target concept, it is possible to pick at most $|X| - 1$ of them such that they are all independent.

We now study some bounds of the VC-dimension of the relationship concept class C_2 of an arbitrary concept class C.

Lemma 5 *Let d be the VC-dimension of a concept class C, then the VC-dimension of the corresponding relationship concept class is at least $d - 1$.*

Proof. The VC-dimension of C is d means that there exists a set of d objects in X that can be shattered by C. Let the objects be $x_1, x_2, ..., x_d$. Then the set of $d - 1$ object pairs $(x_1, x_2), (x_1, x_3), ..., (x_1, x_d)$ can be shattered by C_2 since each relationship assignment A (assigning each of the pairs as a must-link or a cannot-link) can be realized by the concept c where $x_1 \in c$, and $x_i \in c \Leftrightarrow A(x_1, x_i)$, for all $i \in \{2, 3, ..., d\}$, i.e., x_i is in c if and only if it is must-linked to x_1. □

This lower bound is tight because for any d, when $|X| = d$ and $|C| = 2^{|X|}$, we have shown that the VC-dimension of C_2 is $|X| - 1$.

We now switch to the upper bound. We first prove a useful lemma.

Lemma 6 *Given a concept class C defined over a set X of objects, let $Y \subseteq (X, X)$ be a set of object pairs and $X' \subseteq X$ be the set of objects in X that participate in at least one of the pairs in Y. If the relationship concept class of C, C_2, can shatter Y, then there must be a concept in C in which at least $\lceil \frac{|Y|}{2} \rceil$ objects in X' are present, and at least $\lceil \frac{|Y|}{2} \rceil$ objects in X' are absent.*

Proof. We prove the lemma by showing that there exists a relationship assignment of Y that can only be satisfied by such a concept. We will make use of the vertex merging procedure introduced in Section 3 as follows. Given the set Y of pairs, we obtain the corresponding set of objects X', and construct the graph $\tilde{G^0} = (X', \emptyset)$. Then we choose a suitable assignment for Y (to be discussed), and treat Y as a set of instance-level relationships obtained from oracle calls. We repeatedly take out a relationship from the assignment at each time step, and merge the vertices of $\tilde{G^t}$ for each must-link, and add an edge between two vertices for each cannot-link. Each relationship taken out must contain some new information, for otherwise some pairs in Y will not be completely independent, and thus C_2 cannot shatter it. By Lemma 1, the resulting graph after the merging procedure contains a number of connected components, each with one or two vertices. For convenience, for connected components that contain a single vertex, we imagine that there are two vertices, but one of them represents zero objects.

Suppose there are n connected components, and the numbers of objects represented by the two vertices in the i-th connected component are q_i and q_i' respectively. Let $s = \sum_{i=1}^{n} min(q_i, q_i')$, then any concept in C that satisfies this assignment must have at least s of the objects in X' present, and at least s of the objects in X' absent. Therefore our goal is to find an assignment such that $s \geq \lceil \frac{|Y|}{2} \rceil$.

To find the assignment, we use X and Y to form the graph $H = (X, Y)$. Notice that we use a new symbol H here, since it is not a relationship graph for we still have not determined which of the edges are must-links and which are cannot-links. H contains exactly n connected components, each corresponding to one of the connected components in $\tilde{G}^{|Y|}$, the final relationship graph based on Y. Let p_i be the number of pairs in the i-th connected component, then the number of objects represented by the two vertices in the connected component is exactly $p_i + 1$, since H is acyclic, again for otherwise some pairs in Y will not be completely independent. Then we arbitrarily color $\lfloor \frac{p_i+1}{2} \rfloor$ of the vertices red, and the remaining vertices blue. Each edge connecting two vertices of the same color is assigned as a must-link, and each edge connecting two vertices of different colors is assigned as a cannot-link.

For each connected component, the merging procedure combines all red vertices into one vertex and all blue vertices into one vertex. Therefore $min(q_i, q_i') = \lfloor \frac{p_i+1}{2} \rfloor$, and thus $s = \sum_{i=1}^{n} \lfloor \frac{p_i+1}{2} \rfloor \geq \lceil \sum_{i=1}^{n} \frac{p_i}{2} \rceil = \lceil \frac{|Y|}{2} \rceil$. □

We can now prove the following upper bound:

Lemma 7 *Given a concept class C, let $m = \min(\max_{c \in C} |c|, \max_{c \in C} |X - c|)$. Then the VC-dimension of the relationship concept class of C is at most $2m$.*

Proof. Let d be the VC-dimension of the relationship concept class of C, $m_1 = \max_{c \in C} |c|$ and $m_2 = \max_{c \in C} |X - c|$. From Lemma 6, we know that $d \leq 2m_1$ and $d \leq 2m_2$, therefore $d \leq 2min(m_1, m_2) = 2m$. □

It turns out that this bound is again a tight one in that there exists a family of concept classes in which the VC-dimension of the relationship concept class is exactly $2m$. Let $X = \{x_1, x_2, ..., x_{|X|}\}$ where $|X|$ is odd and C be the concept class that includes all subsets of X of size m or less, where $m \leq \lfloor \frac{|X|}{2} \rfloor$. Then m equals $\min(\max_{c \in C} |c|, \max_{c \in C} |X - c|)$. Let C_2 be the relationship concept class of C. The VC-dimension of C_2 is $2m$, since it can shatter the following $2m$-subset of (X, X): $\{(x_1, x_2), (x_1, x_3), ..., (x_1, x_{2m+1})\}$. This is because for any

assignment taken as input knowledge, the final relationship graph produced by the merging procedure consists of two vertices, the smaller of which contains at most m objects. Therefore this assignment is satisfied by the concept of C that includes all the objects in the smaller vertex but none in the larger vertex.

We summarize the results in the following theorem.

Theorem 4 *Given a concept class C of VC-dimension d, let $m = \min(\max_{c\in C} |c|, \max_{c\in C} |X - c|)$, C_2 be the relationship concept class of C and d_2 be its VC-dimension. Then $d-1 \leq d_2 \leq 2m$. Furthermore, there exist concept classes for which $d - 1 = d_2$, and there exist concept classes for which $d_2 = 2m$.*

So far we have been implicitly assuming that X is finite. A nice thing about the above theorem is that the VC-dimension of C_2 does not directly depend on $|X|$, such that as long as all concepts in C have a finite size, the VC-dimension of C_2 is finite, even if $|X|$ is infinite. However, according to Property 11, if $|X|$ is infinite, every concept in C_2 is infinite. This is because there is a must-link between every pair of the infinite number of objects not in the target concept. In this situation, if C is finite, and all concepts in C are finite, then any target concept $c \in C$ can be trivially learnt by enumerating all concepts in C. This guarantees that the corresponding relationship concept $f(c)$ must also be learnable by enumerating a finite number of concepts, albeit each of them is infinite. Therefore we need a finite representation for the argument of the $EM_{f(c)}$ calls. For each concept $f(c)$ in C_2, c is precisely a finite representation that can uniquely determine $f(c)$. If we need to make $EM_{f(c)}$ calls with an argument c'_2 that is not in C_2, there is a finite representation c' such that $c'_2 = f(c')$ if and only if c'_2 involves a finite number of objects in X, and the pairs in c'_2 form two cliques (one of them can be empty). Otherwise, c'_2 may or may not have a finite representation.

For example, the halving algorithm [3] that minimizes the worst case number of EM calls by reducing the version space by half after each call may need to use some concepts c'_2 that has no corresponding concepts $c \in C$ such that $c'_2 = f(c)$. To illustrate, suppose X is the infinite set $\{x_1, x_2, ...\}$ and C contains only three concepts: $\{x_1, x_2\}$, $\{x_1, x_3\}$ and $\{x_1, x_2, x_3\}$. Then the pair (x_1, x_2) receives two votes, (x_1, x_3) receives two votes, but (x_2, x_3) receives only one vote. Therefore, the algorithm will make an EM call with the argument containing the first two pairs but not the third, which violates the "odd number of must-links" rule and is thus inconsistent with all concepts in C.

6 Passive learning

In this section we study the problem of learning c passively, i.e., using PR calls. We will focus on the problem of learning E, the set of must-links of the target concept c. We will also focus on concept classes that are defined over a finite set of objects. Since we have no control of the relationships returned by PR, an adversary can easily prohibit a learner from learning E by returning the same relationship again and again, if the relationship is consistent with at least two concepts in the concept class. It is therefore unreasonable to ask a learner to completely identify E if there are no constraints on the information returned.

We are interested in two problems. First, we want to know what constraints need to be imposed on PR so that E can be completely learnt, and the corresponding learning cost. Second, if the goal is to learn a set of relationships that is sufficiently similar to E according

to a certain measure (e.g. based on the PAC model) rather than to identify E completely, we want to study the number of oracle calls required.

We first study the necessary and sufficient conditions for learning E completely by making PR calls only. After making a sequence of oracle calls, we say that two objects $x, x' \in X$ are directly oracle-connected if $((x, x'), r)$ was either 1) the argument of one or more AR calls, 2) the response of one or more PR calls, or 3) the counter example of one or more ER calls. Two objects $x, x' \in X$ are oracle-connected if there exists a chain of objects such that x is the first object of the chain, x' is the last object, and every two consecutive objects in the chain are directly oracle-connected. If we define binary operators *odc* and *oc* that take two objects as inputs and return true if and only if the two objects are directly oracle-connected and oracle-connected respectively, then both operators are symmetric, and *oc* is transitive. We now discuss the necessary and sufficient conditions in terms of oracle-connectedness.

Lemma 8 *Let $C = 2^X$ be the concept class that contains all subsets of the set X of objects, and $c \in C$ is a target concept. The relationship graph of c can be learnt from PR calls alone if and only if all objects in X are oracle-connected after the calls.*

Proof. The condition is sufficient because for any two objects $x, x' \in X$, suppose they are connected through the chain $x, x_1, x_2, ..., x_n, x'$, then the relationship between x and x_1 and between x_1 and x_2 imply the relationship between x and x_2. Then the relationship between x and $x_i (i = 3..n)$ can be inferred, and finally the relationship between x and x'.

The condition is necessary because if two objects $x_1, x_2 \in X$ are not oracle-connected, the relationship between the two objects is not directly supplied by the oracle calls. Also, after all possible inference of relationships, there exists no object x_3 such that both the relationships between x_1 and x_3 and between x_2 and x_3 are known. Therefore none of the inference rules can infer the relationship between x_1 and x_3, and thus the version space contains more than one relationship graph. □

Corollary 2 *Let $C = 2^X$ be the concept class that contains all subsets of the set X of objects, and $c \in C$ be a target concept. Suppose the examples returned by PR calls are drawn randomly and independently from a distribution over all pairs of distinct objects in X, then the relationship graph of c can be learnt from PR calls alone if and only if there does not exist a set $U \subset X$ such that $\forall x_1 \in U, \forall x_2 \in X - U, Pr(x_1, x_2) = 0$, where $Pr(x_1, x_2)$ is the probability for the pair (x_1, x_2) to be drawn as the example of a PR call.*

We now present a more general result for arbitrary concept classes.

Lemma 9 *Let C be a concept class defined over a set X of objects, and c_i, c_j be two concepts in C. Let P be the set of all object pairs (x, x') such that $AR_{c_i}(x, x') \neq AR_{c_j}(x, x')$. The two concepts can be distinguished by PR calls alone if and only if there exists a pair in P with non-zero probability.*

Proof. The condition is sufficient because if there is a non-zero probability of drawing any object in P, then the two concepts can be distinguished when the pair is drawn.

The condition is necessary because if all pairs in P have zero probability, then they will never be drawn. At the same time, they cannot be inferred by other pairs being drawn because in order to infer different relationships for the same pair of objects, the two concepts must

disagree on the relationship between at least one of the pairs used by the inference, which must also be in P and thus have zero probability of being drawn. □

Therefore we have the following theorem.

Theorem 5 *Let C be a concept class defined over a set X of objects. For any two concepts c_i, c_j, let $P_{i,j}$ be the set of all object pairs (x, x') such that $AR_{c_i}(x, x') \neq AR_{c_j}(x, x')$. Then the relationship graph of c_i can be learnt from PR calls alone if and only if $\forall j s.t. c_i \neq X - c_j, \exists (x, x') \in P_{i,j} s.t. Pr(x, x') > 0$. Therefore, the relationship graphs of all concepts in C can be learnt from PR calls alone if and only if $\forall i, j s.t. c_i \neq X - c_j, \exists (x, x') \in P_{i,j} s.t. Pr(x, x') > 0$.*

The analogous theorem for learning from PM calls is that all concepts in a concept class can be learnt if and only if for every pair of concepts, there exists at least one object the membership of which the two concepts disagree on has a non-zero probability of being drawn. Since every two concepts differ by $O(|X|)$ memberships, but $O(|X|^2)$ relationships, the constraints on the probability distribution of PR examples are much weaker than the constraints on that of PM examples.

Notice that Theorem 5 can be extended to algorithms that involve AR, if the AR oracle is allowed to answer "I don't know" to some queries. Each relationship that the AR oracle cannot answer is analogous to a relationship that the PR oracle will never return (i.e., one with zero probability of being drawn). Therefore if such relationships partition the relationship graph into two or more disjoint partitions, E can never be learnt completely. On the other hand, if E can be learnt, then the fact that AR cannot answer some queries has minimal impact to the learning algorithm presented in Section 3. The total number of oracle calls is only $|X| - 1$ plus the number of times that AR answers "I don't know".

We now switch to the problem of learning an approximation of E that is sufficiently good according to a certain measure.

First, we consider the PAC setting in which the goal is to learn a concept that with a high probability, is approximately correct. More precisely, suppose the real concept is c and the learnt concept is c', we want the following inequality to hold with probability at least $1 - \delta$: $Pr(AM_c(x) \neq AM_{c'}(x)) \leq \epsilon$, where x is an object drawn randomly according to a distribution over X, and δ and ϵ are two quality parameters.

Blumer et al. [9] proved general lower and upper bounds for PAC learning based on the VC-dimension of a concept class. Let d be the VC-dimension of the concept class C, they proved that

- For $0 < \epsilon < 1$, any concept in C can be learnt from $\max(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8d}{\epsilon} \log \frac{13}{\epsilon})$ or more PM calls.
- For $0 < \epsilon < \frac{1}{2}$, no concepts in C can be learnt from $\max(\frac{1-\epsilon}{\epsilon} \ln \frac{1}{\delta}, d(1 - 2(\epsilon(1 - \delta) + \delta)))$ or fewer PM calls.

Let $m = \min(\max_{c \in C} |c|, \max_{c \in C} |X - c|)$, then using these results, and the bounds we obtained in Theorem 4, we can give the bounds for learning concepts in C using PR calls:

- For $0 < \epsilon < 1$, any concept in C can be learnt from $\max(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{16m}{\epsilon} \log \frac{13}{\epsilon})$ or more PR calls.

- For $0 < \epsilon < \frac{1}{2}$, no concepts in C can be learnt from $\max(\frac{1-\epsilon}{\epsilon} \ln \frac{1}{\delta}, (d-1)(1-2(\epsilon(1-\delta)+\delta)))$ or fewer PR calls.

Another way to deal with the problem of approximate learning is to maximize the expected total net reward of oracle calls, which is based on the cost and reward of oracle calls. We will consider this approach in the next section.

7 Mixed learning

We now consider learning E using all three relationship oracles, AR, ER and PR. We will base our discussion on the cost and reward of oracle calls. We assume that each type of oracle call is associated with a constant positive cost. The costs for making an AR, ER and PR call are denoted as λ_A, λ_E and λ_P respectively. The response of each oracle call is given a reward value based on the amount of new information it carries. Suppose an oracle call returns the relationship between a pair of objects (x, x'), we propose three different reward functions, Υ_1, Υ_2 and Υ_3:

- $\Upsilon_1(x, x') = 1$ if the relationship between x and x' was not already known from the relationships contained in or inferred from the responses of previous oracle calls, $\Upsilon_1(x, x') = 0$ otherwise.
- $\Upsilon_2(x, x')$ equals the number of new relationships being known due to the relationship between x and x', assuming all known relationships are contained in or inferred from responses of oracle calls.
- $\Upsilon_3(x, x')$ equals the number of relationship graphs in the version space being pruned by the relationship between x and x'.

The first two reward functions are for general analyses, in which we make no assumptions of the concept class C. Υ_1 is more suitable when the ultimate goal is to learn the target concept c, while Υ_2 is more suitable when the goal is to learn as many of the relationships of the object pairs as possible. Υ_3 applies to situations in which we have some prior knowledge (bias) about the concept class, which can infer relationships without explicit oracle calls. For example, suppose we have got the relationships between objects x_1 and x_2, and between x_1 and x_3. If we have no prior knowledge about C, no matter what the actual relationships are, we get only one additional inferred relationship, the one between x_2 and x_3. However, if we know that C is the set of all $(|X|-1)$-subsets of X, then if both relationships are cannot-links, we immediately know that $c = X - \{x_1\}$. But if both relationships are must-links, we can only prune away three concepts from C. This example shows that when we have some prior knowledge about C, Υ_3 could be a more suitable reward measure than Υ_1 and Υ_2.

It can be seen that for any given learning process, there is a maximum total reward of all oracle calls, which is equal to $|X| - 1$, $\frac{1}{2}|X|(|X| - 1)$ and $|\{f(c) : c \in C\}| - 1$ for Υ_1, Υ_2 and Υ_3 respectively, where f is the function that maps a concept to the set of must-links defined in Section 5. At any time, if an ER call returns true, the reward of the call equals the maximum total reward minus the total reward of all the oracle calls being made so far.

We now setup a probabilistic framework for analyzing the cost and reward of oracle calls. We will assume X to be finite. The responses of PR calls are drawn randomly and independently

from a certain distribution. We denote the probability for PR to return the relationship between the pair (x_1, x_2) as $Pr(PR = (x_1, x_2))$. Similarly, the counter examples of ER calls are drawn randomly and independently from a certain distribution given the guess c'. The probability for ER to return the relationship between the pair $(x1, x2)$ as counter example is denoted as $Pr(ER = (x_1, x_2)|c')$. We assume the learner has a prior probability distribution over the set of all possible concepts of C, where the prior probability for concept c is denoted as $Pr(c)$.

We first study PR. Suppose at a certain time, we receive the relationship between a pair (x, x') of objects through a PR call. Suppose there are k sets of distinct oracle-connected objects $S_1, S_2, ..., S_k$, then the probability that $\Upsilon_1(x, x') = 1$ is $p = 1 - \sum_{i=1}^{k} \sum_{y,y' \in S_i} Pr(PR = (y, y'))$, which is also the expected value of $\Upsilon_1(x, x')$.

For $\Upsilon_2(x, x')$, there are four cases:

- If both x and x' are in the same object set, $\Upsilon_2(x, x') = 0$. The probability is $\sum_{i=1}^{k} \sum_{y,y' \in S_i} Pr(PR = (y, y'))$.
- If both x and x' are not in any of the k object sets, $\Upsilon_2(x, x') = 1$. Let $\tilde{S} = X - \cup_{i=1}^{k} S_i$, then the probability is $\sum_{y,y' \in \tilde{S}} Pr(PR = (y, y'))$.
- If x is in the i-th object set, but x' is not in any of the object sets, then $\Upsilon_2(x, x') = |S_i|$. Reusing the definition of $\tilde{S}$ in the previous case, the probability is $\sum_{y \in S_i} \sum_{y' \in \tilde{S}} Pr(PR = (y, y'))$.
- If x is in the i-th object set, and x' is in the j-th object set ($i \neq j$), then $\Upsilon_2(x, x') = |S_i||S_j|$. The probability is $\sum_{y \in S_i} \sum_{y' \in S_j} Pr(PR = (y, y'))$.

The value of $\Upsilon_3(x, x')$ depends on both the oracle-connected objects and the relationship graphs in the current version space. Suppose the concept being guessed is c', there are two cases:

- If both x and x' are in the i-th oracle-connected object set, then $\Upsilon_3(x, x') = 0$. The probability is $\sum_{i=1}^{k} \sum_{y,y' \in S_i} Pr(ER = (y, y')|c')$.
- Otherwise, the relationship is a new piece of information, and the value of $\Upsilon_3(x, x')$ is equal to the number of concepts that assigns the opposite relationship to the object pair. We can treat each relationship graph in the version space as a higher order object, and two relationship graphs are must-linked if they assign the same relationship to the object pair (x, x'). Otherwise, they are cannot-linked. Using the vertex merging procedure again, the higher order relationship graph of the version space has two vertices, each representing a set of relationship graphs of the original concepts. If all the relationship graphs in the version space assigns the same relationship to the object pair (x, x'), one of the vertices is empty. Recall the notation $P_{i,j}$ as the set of all object pairs the relationships of which the concepts c_i and c_j disagree on, and denote C^t as the current version space, then the probability for $\Upsilon_3(x, x') = 0$ is $\sum_{c_i, c_j \in C^t} \sum_{y,y' \in X : (\neg \exists l s.t. y \in S_l \wedge y' \in S_l) \wedge ((y,y') \notin P_{i,j})} Pr(ER = (y, y')|c')$, where y and y' are two objects in X such that 1) are not in the same oracle-connected object set (i.e., the relationship between them is a new piece of information); and 2) the concepts c_1 and c_2 agree on the relationship between them. The probabilities for other values of $\Upsilon_3(x, x')$ can be determined by looking at the exact content of the relationship graphs in C^t, or lower and upper bounds can be computed based on the sets S_l's and $P_{i,j}$'s.

For AR, the reward of each call depends on the response as in the case of PR, but the probability for receiving each response depends on the nature of the learning algorithm, which can be deterministic. For ER, the reward of each call can be analyzed in almost the same way as in PR, but we also need to consider the probability for an ER call to return true. The probability for that is $Pr(c'|G^t) = \alpha Pr(c')Pr(G^t|c')$, where G^t is the part of the relationship graph learnt so far, and α is the normalization factor of Bayesian inference.

The net reward of a learning process is the total reward of all the oracle calls minus the total cost. The goal of mixed learning is to design the sequence of oracle calls such that the expected net reward is maximized.

Unless we want to explore the probability distribution of the responses of PR calls, for purposes like estimating the testing distribution, it is usually easier to learn using AR calls than PR calls because in the former case we have control of what relationship to ask for but in the latter case we do not. Therefore we will assume that the cost of an AR call is higher than that of a PR call. This makes sense in many practical situations. For example, when we try to study the interactions between different proteins, a PR call could correspond to looking up the literature to identify a known interaction, while an AR call could be a real laboratory experiment, which is much more costly.

Similarly, if the expected reward of an ER call is always smaller than that of an AR call, there is no reason to make ER calls if they have higher cost than an equal number of AR calls. Unlike what we have just said for AR and PR, in reality it does happen that the cost of an ER call is much higher than an AR call. In fact, when a real ER oracle is not available, one way to simulate a stochastic ER is to use a number of AR calls [3]. Therefore unless there is a good probability that we can make ER calls that return true, it is generally more preferable to use AR calls than ER calls.

We now study several algorithms for learning the edges E in the relationship graph of the target concept using AR and PR calls only. The first one is a simple greedy algorithm that at each time step, it always makes the oracle call with the maximum expected net reward. This algorithm is not optimal, as illustrated by the following example. Suppose we want to learn the relationship between a pair of objects. The probability for a PR call to return the relationship is p, and its reward is υ. We have two choices: either to make an AR call to get the relationship, or make PR calls until we receive the relationship. It can be easily verified that all call sequences that involve both AR and PR must have a total net reward not more than one of these two choices. The net reward of using the AR call is $\upsilon - \lambda_A$, while the expected net reward of a PR call is $p\upsilon - \lambda_P$. Therefore the algorithm would choose to make the AR call if $p\upsilon - \lambda_P < \upsilon - \lambda_A$. However, the total expected net reward of using PR calls is $\upsilon - \sum_{i=1}^{\infty}(1-p)^{i-1}pi\lambda_P = \upsilon - \frac{\lambda_P}{p} = \frac{1}{p}(p\upsilon - \lambda_P)$. Therefore the call sequence would be not optimal if $\frac{1}{p}(p\upsilon - \lambda_P) > \upsilon - \lambda_A$. For instance, when $\lambda_A = 5, \lambda_P = 2, \upsilon = 8$ and $p = 0.5$, both inequalities are satisfied and thus the algorithm is not optimal.

On the other extreme, we could always obtain an optimal solution by performing an exhaustive search of a state space. In this approach, each state corresponds to one of the subsets of E. Each state can reach a set of states by performing an action: an AR call or a PR call. If an AR call is made, the transition is deterministic. If a PR call is made, the transition is probabilistic. Each transition is associated with a reward value according to the reward function being used. The expected utility of an action is equal to the expected utility of the state being transited to, plus the expected reward of the transition, minus the cost of the action. The expected utility

of a state is equal to the maximum of the expected utility of all the actions that can be taken in that state, or zero if all actions have negative utility. At any state, the action to be taken is the one with the highest utility.

The above setting is a special form of Markov decision process. There is a finite number of states with a partial order between them based on the number of edges in E already learnt. A state can only move to another state with the number of edges learnt not less than the current state. The learning stops when the final state is reached, which corresponds to the state with all edges in E learnt. At each time step, the learning algorithm can choose to have the next transition being deterministic or probabilistic, both with a fixed cost, but the former one has a fixed reward while the reward for the latter one is probabilistic.

Since we are considering a finite X, the number of states is finite. This means it is possible to setup a finite system of equations for estimating the utility values, for example by using iterative methods. Unfortunately, this is computationally intractable due to the enormous size of the state space. In practice, one may want to adopt an intermediate approach between the two extremes. For example, one possible heuristic is to consider all possible AR calls and $\lfloor \frac{\lambda_A}{\lambda_P} \rfloor$ consecutive PR calls, and take the action with the highest expected net reward.

8 Learning in a multi-concept environment

In this section we extend the learning problem to a multi-concept environment. We assume there is a set of concepts $\mathbf{c} = \{c_1, c_2, ..., c_k\}$, where k is a known integer. We first consider disjoint concepts, in which each object belongs to one and only one concept. There is a must-link between two objects if they are in the same concept. Otherwise, there is a cannot-link between them.

For simplicity, we will assume that all k concepts are non-empty. The relationship graph of $\mathbf{c}$ contains k cliques, and its complement is a k-partite graph, with every vertex connecting to all vertices in other parts.

Among the three inference rules, the transitivity of must-links (Property 3) and the propagation of cannot-links (Property 4) still hold, but the elimination of cannot-links (Property 5) does not hold anymore. This rule can be generalized to hold for an arbitrary number of concepts as follows:

Property 12 Elimination of cannot-links (for k concepts):
For any objects $x_1, x_2, ..., x_{k+1} \in X$, if 1) for all distinct $i, j \in \{1, 2, ..., k\}$ there is a cannot link between x_i and x_j, and 2) for all $i \in \{2, ..., k\}$ there is a cannot-link between x_i and x_{k+1}, then there is a must-link between x_1 and x_{k+1}.

In other words, if we know that $x_1, x_2, ..., x_k$ all belong to different concepts, and x_{k+1} belongs to a concept different from the ones to which $x_2, x_3, ..., x_k$ belong, then x_{k+1} must belong to the concept to which x_1 belongs.

This generalization has a serious consequence to the general learning algorithm that we discussed. The basic impact is that it is now possible to receive more than $O(|X|)$ distinct relationships from oracles without being able to infer even one new relationship. Consider the following example. Suppose $|X| = n(k-1)$, where n is a positive integer. We partition X into $k-1$ groups, each with n objects. Suppose we receive from oracle calls a cannot-link for each inter-group object pairs. In total there are $\frac{|X|n(k-2)}{2} = \frac{|X|^2(1-\frac{2}{k})}{2}$ of them. Among all the $|X|$

objects, we cannot find any set of k objects such that we have received a relationship between every pair of objects within the set. This can be seen in two steps. First, since all relationships are between objects of different groups, the k objects must all come from different groups in order to have all pairs with a relationship received. But there are only $k-1$ groups, so it is impossible to pick k objects all from different groups.

Now, since the inference rule in Property 12 requires a group of k fully connected objects, it cannot be applied to the example scenario. In addition, both of the other two inference rules require at least one must-link, but we have not received any must-links in the example, so they also cannot be applied. In other words, even we have received $\frac{|X|^2(1-\frac{2}{k})}{2}$ relationships from oracle calls, we still cannot make any inference at all.

The number $\frac{|X|^2(1-\frac{2}{k})}{2}$ is a lower bound of the number of distinct oracle calls required before we can make the first inference in the worst case, i.e., when we do not have control of what oracle calls to make. There is a big gap of the exact value of the number when $k=2$ and $k=3$. When $k=2$, this number is equal to 0. And in fact, we know that if any two object pairs among a set of three objects have their relationships specified, we can infer the third one. That means after receiving $\frac{|X|}{2} = O(|X|)$ relationships, we must be able to make an inference. However, when $k>2$, the formula is of order $\Theta(|X|^2)$. Therefore, if we do not control the sequence of oracle calls, in the worst case it takes $\Omega(|X|^2)$ calls to completely learn the set of edges E of the relationship graph of the target concept, even we receive some new information in each call. Comparing to the $|X|-1$ calls guarantee when $k=2$, the increase in the number of oracle calls is substantial.

Fortunately, by controlling the call sequence, it is possible to learn E using $O(|X|)$ AR calls. One possible way is as follows: we choose an object, and use AR to query the relationship between it and every other object. The object and all objects must-linked to it form a concept. Then we choose another object, and use AR to query the relationship between it and every object not in the first concept. We repeat the process until we have formed $k-1$ concepts. The remaining objects then belong to the last concept. The worst case appears when the relationships received in the first $k-2$ scans are all cannot-links. In that case, the total number of oracle calls needed is $(|X|-1)+(|X|-2)+...+(|X|-k+1) = (k-1)(|X|-\frac{k}{2})$. When $|X| \gg k$, this number is close to $|X-1|(k-1)$, which seems not too desirable since it is the number of oracle calls required for learning $k-1$ concepts separately using the individual AR oracles of the concepts.

Unfortunately, it is impossible to use a smaller number of oracle calls in the worst case. To completely construct E, we need to receive and/or infer at least $|X|-k$ different must-links in total. If all received relationships are cannot-links, each must-link must be inferred by Property 12. Each such inference involves a set of k objects all cannot-linked to each other, plus an object that is cannot-linked to $k-1$ of them. These $k+1$ objects can be viewed as a set of objects with all but one pair of objects not cannot-linked together. The must-link being inferred is exactly between these two objects. If we view the objects in X as vertices and cannot-links as edges, we want to solve the following graph problem: in a graph with $|X|$ vertices, what is the minimum number of edges required to create $|X|-k$ "defective cliques" of size $k+1$, each with exactly one missing edge, and no two of them share the same missing edge?

We now prove that this number is exactly $(k-1)(|X|-\frac{k}{2})$. Suppose there is a graph that contains at least $|X|-k$ defective cliques each with exactly one missing edge and no

two of them share the same missing edge. We pick any two of them. Suppose they involve objects $\{x_1, x_2, ..., x_{k+1}\}$ and $\{x'_1, x'_2, ..., x'_{k+1}\}$ respectively, where the missing edge in the first defective clique is (x_1, x_2), and the one in the second defective clique is (x'_1, x'_2). Then first we know that at least one of the objects in (x'_1, x'_2) must be different from both objects in (x_1, x_2), for otherwise the two defective cliques would have the same missing edge. Also, we know that there must not exist integers $i, j (1 \leq i \leq k+1, 3 \leq j \leq k+1)$ such that $(x_i, x_j) = (x'_1, x'_2)$, since this edge would be missing in the first defective clique but present in the second one, which is impossible. Therefore, at least one of x'_1, x'_2 is not in $\{x_1, x_2, ..., x_{k+1}\}$. This means the second defective clique contains at least $k-1$ edges that are absent in the first defective clique.

Then we pick a third defective clique. Using the same argument again, at least one of the two objects involved in the missing edge must not be involved in both the first and the second defective cliques. Therefore this third defective clique again must contain at least $k-1$ edges that are absent in the first two defective cliques. By repeating the process to examine $|X| - k$ defective cliques, we notice that the graph must contain at least $\frac{k(k+1)}{2} - 1$ edges that define the first defective clique, and at least $k-1$ new edges for each of the other $|X|-k-1$ defective cliques. Summing up, the graph must contain at least $\frac{k(k+1)}{2} - 1 + (k-1)(|X|-k-1) = (k-1)(|X| - \frac{k}{2})$.

Based on the result, we can now state the following theorem.

Theorem 6 *In order to completely learn the relationship graph of k disjoint non-empty concepts defined over a set X of objects using AR, ER and PR calls only, in the worst case at least $(k-1)(|X| - \frac{k}{2})$ oracle calls are needed. This lower bound is tight because there exists an algorithm that uses AR calls only, which learns the relationship graph using exactly $(k-1)(|X| - \frac{k}{2})$ calls in the worst case.*

As in the two-concept case, after learning E, we know the exact concepts if we have an example of $k-1$ of the concepts, which can be obtained by making $k-1$ AM calls.

If we use AM directly to learn k disjoint concepts, obviously we need to make at most $|X|$ calls. This means the information carried by a membership is approximately equal to $k-1$ times the information carried by a relationship.

For passive learning, in order to learn the relationship graph of k disjoint concepts completely, we need to be able to receive and/or infer all must-links. The sufficient and necessary conditions are the same: if we treat the objects as vertices and must-links as edges, we need to be able to receive and/or infer must-links so that all the objects in each concept are connected.

We now switch to the case when the concepts are not disjoint, i.e., each object may belong to zero, one or more concepts in $\mathbf{c}$. For an object $x \in X$, we will use $c(x)$ to denote the set of concepts that x belongs to, among the k concepts under consideration. The relationship between two objects is thus no longer the relationship between the two concepts that they belong to, but the two sets of concepts that they belong to. For example, based on the oracles that supply language-level information defined by Angluin [3], we can define various kinds of active oracles that supply instance-level information in a non-disjoint multi-concept environment, each of which being a binary function that takes two objects as inputs, and returns a truth value:

- Identity[5]: $AR^{=}(x_1, x_2) \Leftrightarrow c(x_1) = c(x_2)$.

[5] In [3], the term "equivalence" is used instead of "identity". We prefer the latter in order not to get confused with the ER oracle.

- Subset: $AR^{\subseteq}(x_1, x_2) \Leftrightarrow c(x_1) \subseteq c(x_2)$.
- Superset: $AR^{\supseteq}(x_1, x_2) \Leftrightarrow c(x_1) \supseteq c(x_2)$.
- Disjointness: $AR^{\emptyset}(x_1, x_2) \Leftrightarrow c(x_1) \cap c(x_2) = \emptyset$.
- Exhaustiveness: $AR^{\mathbf{c}}(x_1, x_2) \Leftrightarrow c(x_1) \cup c(x_2) = \mathbf{c}$.

We can define passive oracles similarly, but we will focus on the active oracles. The symbols used in the oracle definitions will also be used to define relationships. For example, $((x_1, x_2), =)$ means $AR^{=}(x_1, x_2)$ is true. Notice that since the subset operator is not commutative, (x_1, x_2) here means an ordered pair, and the first object is always used as the first argument of oracle calls.

There are some inference rules for the relationships:

1. $((x_1, x_2), ?) \Rightarrow ((x_2, x_1), ?)$, where $? \in \{=, \emptyset, \mathbf{c}\}$.
2. $((x_1, x_2), \subseteq) \Leftrightarrow ((x_2, x_1), \supseteq)$.
3. $((x_1, x_2), =) \wedge ((x_2, x_3), ?) \Rightarrow ((x_1, x_3), ?)$, where $? \in \{=, \subseteq, \supseteq, \emptyset, \mathbf{c}\}$.
4. $\neg((x_1, x_2), =) \wedge ((x_1, x_2), ?) \Rightarrow \neg((x_2, x_1), ?)$, where $? \in \{\subseteq, \supseteq\}$.
5. $((x_1, x_2), ?) \wedge ((x_2, x_1), ?) \Rightarrow ((x_1, x_2), =)$, where $? \in \{\subseteq, \supseteq\}$.
6. $((x_1, x_2), \subseteq) \wedge ((x_2, x_3), ?) \Rightarrow ((x_1, x_3), ?)$, where $? \in \{\subseteq, \emptyset\}$.
7. $((x_1, x_2), \supseteq) \wedge ((x_2, x_3), ?) \Rightarrow ((x_1, x_3), ?)$, where $? \in \{\supseteq, \mathbf{c}\}$.

We first study each oracle individually, to see what can be learnt if we use only one kind of oracle calls. Using $AR^{=}$ calls only, we can partition the objects into disjoint groups, such that all objects within a group belong to exactly the same subset of concepts in $\mathbf{c}$. We can perform some analysis by setting up a link between disjoint and non-disjoint concepts. If we take each of the 2^k subsets of $\mathbf{c}$ as a higher-order concept, then each object belongs to exactly one higher-order concept, i.e., the higher-order concepts are disjoint. The $AR^{=}$ oracle in the non-disjoint environment becomes exactly the AR oracle of the disjoint higher-order concepts. Then by Theorem 6, we know that if all the higher-order concepts are non-empty, the groups can be learnt by using $(2^k - 1)(|X| - 2^{k-1})$ oracle calls. This is also the minimum number of oracle calls required in the worst case. If some of the higher-order concepts are empty, then the groups can be learnt by making $AR^{=}$ calls between every pair of objects, which requires $\frac{|X|(|X|-1)}{2}$ oracle calls. The scanning algorithm described previously for learning disjoint concepts guarantees that the actual number of oracle calls is the minimum of the two.

For $AR^{\subseteq}$, we can learn something more. First, because each $AR^{=}$ can be simulated by two $AR^{\subseteq}$ calls using Rule 5, we can also learn the object groups that can be learnt by using $AR^{=}$ calls. In addition, we can also learn a partial order of the groups, based on the subset relationships between the different subsets of concepts represented by the groups. If there are 2^k different groups, then we know the complete subset relationships between the different groups. This means, for example, we know how many concepts each object in X belongs to. On the other hand, if there are fewer than 2^k groups, suppose we view the partial order as a lattice with each group as a node and groups representing larger subsets of concepts on the top, then

the lower bound of the number of concepts of a group is equal to the number of edges in the longest path that goes down the lattice from the group, and the upper bound is equal to k minus the number of edges in the longest path that goes up the lattice from the group. $AR^{\supseteq}$ can achieve exactly the same thing.

For $AR^{\emptyset}$, again we can obtain some information about the number of concepts each object belongs to. First, all objects that do not belong to any concept can be detected, because they are the only objects x that the call $AR^{\emptyset}(x, x)$ returns true. Then for the remaining objects, we put them into groups such that 1) for every two objects x_1, x_2 in the same group, $AR^{\emptyset}(x_1, x_2)$ is false; and 2) for every two objects x_1, x_2 in the same group and an object x' in a different group, $AR^{\emptyset}(x_1, x') = AR^{\emptyset}(x_2, x')$. The objects that do not belong to any concept are also put into a group. If there are exactly 2^k groups, then each group corresponds to exactly one distinct subset of $\mathbf{c}$. It is impossible to know exactly which subsets the groups represent, but it is possible to determine the size of the subsets represented by each group. For a subset of size n, there are exactly 2^{k-n} subsets of $\mathbf{c}$ that are disjoint from it. Therefore if the objects of a group form disjointness-links with objects of 2^{k-n} different groups, then the group represents a subset of n different concepts in $\mathbf{c}$. The subset information between different groups can also be determined. For a group that represents a subset $\mathbf{c'}$ of size n, we look for all groups that represent a subset $\mathbf{c''}$ of size $n+1$. $\mathbf{c'}$ is a subset of $\mathbf{c''}$ if and only if whenever $\mathbf{c''}$ is disjoint from a subset of concepts, $\mathbf{c'}$ is also disjoint from it. Therefore, if all 2^k groups are formed, we can use $AR^{\emptyset}$ calls to learn the same information we described that can be learnt form $AR^{\subseteq}$ calls.

On the other hand, if there are fewer than 2^k groups, then objects in the same group may belong to different subsets of concepts. In this case, the number of disjoint groups can be used to obtain an upper bound of the number of concepts each object of a group must belong to. If an object forms disjointness-links with the objects of n other groups, then it must belong to at most $k - \lceil \log_2 n \rceil$ different concepts. In addition, we also obtain some extra information. Suppose there is a group in which there is an object x_i with $c(x_i) = \mathbf{c_i}$ and there is an object x_j with $c(x_j) = \mathbf{c_j}$, where $\mathbf{c_i}$ and $\mathbf{c_j}$ are two different subsets of concepts. The symmetric difference between them is the set $(\mathbf{c_i} - \mathbf{c_j}) \cup (\mathbf{c_j} - \mathbf{c_i})$. Then any object in X that belongs to a concept in the symmetric difference must also belong to at least one concept in $\mathbf{c_i} \cup \mathbf{c_j}$, for otherwise the object can be used to distinguish $\mathbf{c_i}$ from $\mathbf{c_j}$, and thus x_i and x_j would have been put into two different groups. Therefore if later we identify $\mathbf{c_i}$ and $\mathbf{c_j}$ completely, we can come up with a list of subsets of concepts that is different from $c(x), \forall x \in X$.

The case for $AR^{\mathbf{c}}$ is similar to that of $AR^{\emptyset}$. We can identify all objects that belong to all the concepts in $\mathbf{c}$, because they are the only objects x that the call $AR^{\mathbf{c}}(x, x)$ returns true. Then we can form groups in a similar fashion. If there are 2^k groups, we can deduce the subset information between different groups completely. Otherwise, we can determine the minimum number of concepts each object belongs to, and obtain some rules that specify which subsets are not equal to any $c(x), \forall x \in X$.

Now, having studied each of the five oracles individually, let us consider the following learning problem. Suppose we also have access to the AR oracles of the k individual concepts, but the cost of making one call to these oracles is much higher than making one call to the five oracles above. We would like to learn the relationship graphs of the k concepts using the smallest number of calls to the k AR oracles.

We first determine the baseline. If we learn the k relationship graphs using the individual AR oracles only, we need to make $|X| - 1$ calls to each oracle, giving a total of $k(|X| - 1)$ calls.

We will briefly study a two-step algorithm. In the first step, we use the five types of oracles

to learn the object groups and the subset information between them. We will again use a lattice to represent the information. The lattice can be learnt by first using $\min((2^k-1)(|X|-2^{k-1}), \frac{|X|(|X|-1)}{2})$ $AR^{=}$ calls to learn the groups. Suppose there are g groups, then the subset information between them can be learnt by using $\frac{g(g-1)}{2}$ $AR^{\subseteq}$ calls. In total, the lattice can be learnt by using at most $|X|(|X|-1)$ calls.

Then in the second step, we use the lattice and calls to the individual AR oracles to learn the k relationship graphs. We first consider a simple case, in which the lattice contains all 2^k nodes. In this case, we do not need to make any calls to the individual AR oracles at all. We simply arbitrarily assign the k concepts to the k nodes that represent one single concept. Then from the lattice structure, the concepts represented by each node are completely determined. The k relationship graphs can then be exactly deduced, subject to permutation of the concepts. Notice that due to symmetry, we could also use the k nodes each of which represents $k-1$ concepts as reference.

There is an interesting remark for this case. Since there is at least one object that does not belong to any concept, we can use this object as a reference to exactly determine which concept each of the k nodes represents. We first use at most $k-1$ AR calls to deduce the concept the first node represents, then use at most $k-2$ calls to deduce the concept the second node represents, and so on. The total number of AR calls required to completely determine the k concepts (not only their relationship graphs) is at most $(k-1)+(k-2)+...+1=\frac{k(k-1)}{2}$.

Next, we consider lattices with fewer than 2^k nodes. In this case, the number of concepts represented by a node and the number of concepts represented by a child of this node may be larger than one. We will first use the lattice structure as well as $AR^{\emptyset}$ and $AR^{\mathbf{c}}$ calls to determine the upper and lower bounds of the number of concepts represented by each node. We will also know exactly which node pairs represent concepts that are disjoint and exhaustive. Then we use the individual AR oracles to learn the k relationship graphs as follows: for each node in the lattice, we pick an object as its representative. Then we pick a longest path that goes up the lattice, and extract their representatives $\{x_1, x_2, ..., x_n\}$, such that for every $i < n$, the number of concepts that x_{i+1} belongs to is at least one more than the number of concepts that x_i belongs to. Then we post AR calls to learn the relationship between all pairs of consecutive objects (x_i, x_{i+1}) in the path with respect to each concept. First we consider x_1 and x_2. We repeatedly makes the call $AR_{c_j}(x_1, x_2)$, for $j = 1, 2, ...$ Since the set of concepts that x_1 belongs to is a subset of the set of concepts that x_2 belongs to, at least one of the AR_{c_j} calls should return false, and for each AR_{c_j} call that returns false, we know that $x_1 \notin c_j$, and $x_i \in c_j$, for all $i > 1$. Also, suppose the lower and upper bound information shows that x_1 belongs to at least n_1 concepts and x_2 belongs to at most n_2 concepts, then we know that there must be at most $n_2 - n_1$ calls that return false. Once we get that number, all the remaining calls between x_1 and x_2 must all return true, and so we do not actually need to make the calls.

After getting all k relationships between x_1 and x_2 with respect to the k concepts, we update the lower bounds of the number of concepts each x_i belongs to, for all $i > 2$, so that each x_{i+1} belongs to at least one more concept than x_i. After the updates, we work on the relationships between x_2 and x_3. The strategy is basically the same as in the case of x_1 and x_2, but now for all $c_j \in \mathbf{c}$ with $AR_{c_j}(x_1, x_2)$ returning false, we know that x_2 and x_3 both belong to this concept, and thus the must-link is already inferred and there is no need to call $AR_{c_j}(x_2, x_3)$.

When all the objects in the path are processed, we update the lower bounds and upper bounds of the number of concepts of all other nodes in the lattice based on the new lower and

upper bounds of the nodes involved in the path. We also propagate the known memberships to other nodes as follows: suppose x_i is one of the objects in the chain, and it represents node i. Then for each ancestor node of node i, the set of concepts that it represents must contain all concepts that x_i was found to belong to. Similarly, all descendent nodes of node i must not represent any concept that x_i was found not to belong to, all nodes disjoint from node i must not contain any concept that x_i was found to belong to, and all nodes having an exhaustive-relationship with node i must contain all concepts that x_i was found not to belong to.

After performing all the inference, if there are still some objects whose relationships with the first path of objects are unknown with respect to some concepts, we pick another path and make AR calls between them as well as between one of the objects in this path and one of the objects in the first path if necessary to fill in the missing relationships. This is repeated until every representative object is oracle-connected to every other representative object with respect to each concept.

The exact number of calls to the individual AR oracles depends on the structure of the lattice as well as the disjointness and exhaustiveness information. In the worst case, where each group contains only one object and all groups do not have any subset, disjointness or exhaustiveness relationships, it takes $k(|X| - 1)$ calls to the individual AR oracles. In general, the more information we receive from the five oracles, the fewer individual AR calls we need to make.

9 Learning from noisy oracles

In this section we study the problem of learning form noisy oracles, i.e., oracles that can make errors. We will focus on one single type of errors, namely returning an incorrect relationship between a pair of objects. We will consider only the disjoint two-concept environment, i.e., each object is either in the target concept or not. We assume that for each relationship received from an oracle (an answer from AR, a response from PR, or a counter example from ER), there is a fixed probability ϵ that it is incorrect. We will augment the notation of a relationship by adding as the third element the posterior probability that it is correct given the relationships obtained from the oracle calls being made so far (the "observed relationships"). For example, the relationship $((x_1, x_2), r)$ with posterior probability p being correct is denoted as $((x_1, x_2), r, p)$.

For each object pair (x_1, x_2), there are two possible relationships $((x_1, x_2), 1, p)$ and $((x_1, x_2), -1, q)$ where p and q are some probabilities. At any time, we require that $p + q = 1$.

To determine the posterior probabilities, we need a prior distribution over the set of all possible relationship graphs. We require the prior to assign zero probability to all inconsistent relationship graphs, i.e., all relationship graphs that violate the "odd number of must-links" rule among any three objects. The prior distribution can be derived from a prior distribution over the set of all membership assignments. If we can assume independence between the memberships of different objects, then the latter distribution can in turn be derived from the individual probabilities for each object to be in the target concept, $Pr(x_i \in c)$. Notice that it is always invalid to assume independence between the real relationships of different object pairs, due to the "odd number of must-links" property. Later we will study a restricted form of independence assumption of the real relationships.

For any two objects x_i and x_j, we will use the symbol r_{ij} to denote the relationship between

them received form an oracle, and the symbol r^*_{ij} to denote the (unknown) real relationship between them.

We first talk about a single relationship. Suppose we obtain a relationship $((x_1, x_2), r_{12})$ from an oracle. The posterior probability that the real relationship is r^*_{12} is as follows:

$$\begin{aligned} Pr(r^*_{12}|r_{12}) &= \frac{Pr(r_{12}|r^*_{12})Pr(r^*_{12})}{Pr(r_{12})} \\ &= \alpha Pr(r_{12}|r^*_{12})Pr(r^*_{12}), \end{aligned}$$

where $Pr(r_{12}|r^*_{12}) = \epsilon^{\frac{1-r_{12}r^*_{12}}{2}}(1-\epsilon)^{\frac{1+r_{12}r^*_{12}}{2}}$, and α is the normalization factor $\frac{1}{Pr(r_{12})}$. We will keep using α as the normalization factor of Bayesian inference, and it can take different values in different equations.

One advantage of using this probabilistic approach is that we can easily handle contradictory relationships and relationships that have been returned from oracle calls multiple times. Suppose we have received $((x_1, x_2), r_{12})$ i times and $((x_1, x_2), -r_{12})$ j times, then the posterior probability that the real relationship is r^*_{12} is as follows:

$$\begin{aligned} Pr(r^*_{12}|observed\_relationships) &= \frac{Pr(observed\_relationships|r^*_{12})Pr(r^*_{12})}{Pr(observed\_relationships)} \\ &= \alpha(Pr(r_{12}|r^*_{12}))^i(1-Pr(r_{12}|r^*_{12}))^j Pr(r^*_{12}) \end{aligned}$$

The second equation follows from the first one because the event r_{12} is conditionally independent of all other observed relationships given r^*_{12}. This is due to the constant error rate of the oracles.

Now we consider a slightly more complicated situation, in which we have made 3 oracle calls and received 3 relationships, which are exactly the 3 relationships between the objects x_1, x_2 and x_3. We want to know the posterior probability $Pr(r^*_{12}, r^*_{13}, r^*_{23}|r_{12}, r_{13}, r_{23})$, where r_{12}, r_{13} and r_{23} are the observed relationships. This can be calculated as follows:

$$\begin{aligned} Pr(r^*_{12}, r^*_{13}, r^*_{23}|r_{12}, r_{13}, r_{23}) &= \frac{Pr(r_{12}, r_{13}, r_{23}|r^*_{12}, r^*_{13}, r^*_{23})Pr(r^*_{12}, r^*_{13}, r^*_{23})}{Pr(r_{12}, r_{13}, r_{23})} \\ &= \alpha Pr(r_{12}|r^*_{12})Pr(r_{13}|r^*_{13})Pr(r_{23}|r^*_{23})Pr(r^*_{12}, r^*_{13}, r^*_{23}) \quad (1) \end{aligned}$$

The posterior probabilities $Pr(r^*_{ij}|r_{12}, r_{13}, r_{23})$ for $1 \le i < j \le 3$ can then be obtained as a marginal probability by summing up the joint probabilities. For example,

$$Pr(r^*_{12}|r_{12}, r_{13}, r_{23}) = \sum_{r^*_{13}\in\{1,-1\}} \sum_{r^*_{23}\in\{1,-1\}} Pr(r^*_{12}, r^*_{13}, r^*_{23}|r_{12}, r_{13}, r_{23}) \quad (2)$$

To illustrate, let us consider a simple example. Suppose the distribution over all membership assignments is uniform, i.e., each of the 8 possible assignments have equal probability of $\frac{1}{8}$. The distribution over all relationship graphs can then be derived as in Table 1.

From the table, it can be seen that the prior distribution of relationship graphs is a uniform distribution over all consistent relationship graphs, each with a probability of $\frac{1}{4}$. In fact, this

Table 1: Distribution of relationship graphs based on a uniform prior of the membership assignments.

x_1	x_2	x_3	r^*_{12}	r^*_{13}	r^*_{23}	Prob.
0	0	0	1	1	1	$\frac{1}{8}$
0	0	1	1	-1	-1	$\frac{1}{8}$
0	1	0	-1	1	-1	$\frac{1}{8}$
0	1	1	-1	-1	1	$\frac{1}{8}$
1	0	0	-1	-1	1	$\frac{1}{8}$
1	0	1	-1	1	-1	$\frac{1}{8}$
1	1	0	1	-1	-1	$\frac{1}{8}$
1	1	1	1	1	1	$\frac{1}{8}$

phenomenon is generally true for any number of objects, since for any two objects, exactly half of the $2^{|X|}$ possible concepts contain both of them or neither of them, and exactly half contain exactly one of them.

Based on this prior, we can calculate the posterior according to the observed data using Equation 1 and Equation 2. The results are shown in Table 2.

Table 2: Posterior distribution of relationships based on a uniform prior of the membership assignments.

r_{12}	r_{13}	r_{23}	$Pr(r^*_{12}=1\|r_{12},r_{13},r_{23})$	$Pr(r^*_{13}=1\|r_{12},r_{13},r_{23})$	$Pr(r^*_{23}=1\|r_{12},r_{13},r_{23})$
1	1	1	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$
1	1	-1	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$
1	-1	1	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$
1	-1	-1	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$
-1	1	1	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{2(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$
-1	1	-1	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$
-1	-1	1	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{2\epsilon^2}{3\epsilon^2+(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{3\epsilon^2+(1-\epsilon)^2}$
-1	-1	-1	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$	$\frac{\epsilon^2+(1-\epsilon)^2}{\epsilon^2+3(1-\epsilon)^2}$

Using exactly the same approach, in theory we can derive the posterior probabilities of all relationships. Let R be the set of relationships received directly from oracle calls (here we assume no relationships have been received multiple times, otherwise the following equations can be modified easily according to our previous discussion on contradictory and repeated relationships), R^* be the set of real relationships of the object pairs in R, and R'^* be the set of real relationships of all other pairs. Then we have

$$\begin{aligned} Pr(R^*, R'^*|R) &= \frac{Pr(R|R^*, R'^*)Pr(R^*, R'^*)}{Pr(R)} \\ &= \alpha Pr(R^*, R'^*) \prod_{r_{ij}\in R, r^*_{ij}\in R^*} Pr(r_{ij}|r^*_{ij}) \end{aligned} \tag{3}$$

where in Equation 3, we use the indices i and j to relate the object pairs in R and R^*.

The posterior probability of any relationship $r^*_{ij} \in R^* \cup R'^*$ can then be obtained by summing up the joint probabilities. Let $R''^* = R^* \cup R'^* - r^*_{ij}$ and $n = |R''^*|$. For convenience, assume R''^* is an ordered set, then

$$Pr(r^*_{ij}|R) = \sum_{R''^* \in \{1,-1\}^n} Pr(r^*_{ij}, R''^*|R) \tag{4}$$

Since all the terms in Equation 3 are known, the posterior probabilities of all relationships in R^* and R'^* can be computed without making any independence assumption of the real relationships.

In practice, however, the above calculations are computationally intractable due to the exponential number of additions in Equation 4.

We will study some ways to reduce the computational cost, from general ones that do not require any assumptions, to specific ones that rely on some special assumptions. We first notice that the product term $\prod_{r_{ij} \in R, r^*_{ij} \in R^*} Pr(r_{ij}|r^*_{ij})$ in Equation 3 depends only on the relationships in R and R^*. Therefore when R^* is kept constant, all probabilities $Pr(R^*, R'^*|R)$ differ only by the prior term $Pr(R^*, R'^*)$. Further, since there are only two possible values for $Pr(r_{ij}|r^*_{ij})$ depending on whether $r_{ij} = r^*_{ij}$, the product term can take only $|R| + 1$ different values, which can be precomputed. These two observations suggest an incremental approach that could save a lot of multiplications. To calculate $Pr(r^*_{ij}|R)$, we keep $|R| + 1$ accumulators, one for each possible value of the product term. Instead of computing each $Pr(r^*_{ij}, R''^*|R)$ separately before summing them up, we add the prior $Pr(r^*_{ij}, R''^*)$ to the corresponding accumulator. After accumulating all the priors, $Pr(r^*_{ij}|R)$ is equal to the sum of the values in the accumulators multiplied by the corresponding value of the product terms.

Next, notice that it is not needed to use Equation 3 and Equation 4 directly to calculate the posterior probability of every relationship. Suppose we have computed the joint probabilities $Pr(r^*_{12}, r^*_{23}|R)$, then we can infer $Pr(r^*_{13}|R)$. For example,

$$\begin{aligned}
Pr(r^*_{13} = 1|R) &= \sum_{r^*_{12} \in \{1,-1\}} \sum_{r^*_{23} \in \{1,-1\}} Pr(r^*_{12}, r^*_{23}, r^*_{13} = 1|R) \\
&= Pr(r^*_{12} = 1, r^*_{23} = 1, r^*_{13} = 1|R) + Pr(r^*_{12} = -1, r^*_{23} = -1, r^*_{13} = 1|R) \\
&= Pr(r^*_{13} = 1|r^*_{12} = 1, r^*_{23} = 1, R) Pr(r^*_{12} = 1, r^*_{23} = 1|R) + \\
&\quad Pr(r^*_{13} = 1|r^*_{12} = -1, r^*_{23} = -1, R) Pr(r^*_{12} = -1, r^*_{23} = -1|R) \\
&= Pr(r^*_{12} = 1, r^*_{23} = 1|R) + Pr(r^*_{12} = -1, r^*_{23} = -1|R)
\end{aligned}$$

Pushing the idea further, it is observed that although there are $2^{O(|X|^2)}$ terms in the summation in Equation 4, only $2^{O(\frac{|X|}{2})}$ of them are non-zero, since all the others correspond to inconsistent relationship graphs.

Third, it is observed that when computing the probabilities $Pr(r^*_{ij}|R)$ for two different pairs of i, j, a lot of intermediate sums are shared by both calculations, which can be reused. Extending the idea, if we need to compute all the posterior probabilities $Pr(r^*_{ij}|R)$, we can do all the computations in a single run as follows. We pick a joint probability $Pr(R^*, R'^*|R)$.

We know that half of the computations require this value, and the other half does not, so we keep two accumulators one with the value and one with zero. Then we pick another joint probability. Again, only half of the computations require this value, so we make one copy of the two original accumulators, and add the probability only to the copy. We repeat this process until we have picked all the joint probabilities. By that time, we will have computed all the posterior probabilities $Pr(r^*_{ij}|R)$.

All the above methods reduce the overall computational time significantly, but the time complexity is still exponential because the computation takes time proportional to the number of joint probabilities. Also, some of the methods require exponential space.

In order to make the calculations computationally tractable, we will make an assumption of the real relationships. Suppose we are given two sets of real relationships R^*_1 and R^*_2, both with a non-zero joint prior probability. Denote S^*_1 and S^*_2 as the sets of object pairs involved in the relationships in the two sets respectively. We assume that R^*_1 and R^*_2 are independent of each other if there does not exist a cycle of distinct objects such that 1) every edge involved is in either S^*_1 or S^*_2; 2) S^*_1 contains at least one of the edges and 3) S^*_2 contains at least one of the edges.

Let us consider some examples. Based on the assumption,

- $\{r^*_{12}\}$ is independent of $\{r^*_{23}\}$ because there exist no cycles that satisfy requirement 1.
- $\{r^*_{12}\}$ may not be independent of $\{r^*_{23}, r^*_{13}\}$ because the cycle (x_1, x_2, x_3) satisfies all three requirements. From this example, we can see that the assumption does not violate the "odd number of must-links" rule.
- $\{r^*_{12}, r^*_{34}\}$ may not be independent of $\{r^*_{14}, r^*_{23}\}$ because the cycle (x_1, x_2, x_3, x_4) satisfies all three requirements.
- $\{r^*_{12}\}$ is independent of $\{r^*_{23}, r^*_{24}, r^*_{34}\}$ because there does not exist a cycle that satisfies all three requirements.

We call this assumption the restricted independence assumption of relationships. The assumption can be interpreted as follows: two sets of relationships are independent if the "odd number of must-links" rule does not create any dependence between them. We first explain why we want to make this assumption, and then discuss a necessary condition for this assumption to hold.

First, we notice that the assumption implies that the joint posterior probability of a set of real relationships R'^* given a set R of observed relationships is equal to its prior if R^* and R'^* are independent, where R^* is the set of real relationships for the object pairs in R:

$$\begin{aligned} Pr(R'^*|R) &= \sum_{R^* \in \{1,-1\}^{|R^*|}} Pr(R'^*, R^*|R) \\ &= \sum_{R^* \in \{1,-1\}^{|R^*|}} \frac{Pr(R|R'^*, R^*)Pr(R'^*, R^*)}{Pr(R)} \\ &= \sum_{R^* \in \{1,-1\}^{|R^*|}} \frac{Pr(R|R^*)Pr(R'^*, R^*)}{Pr(R)} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{R^* \in \{1,-1\}^{|R^*|}} \frac{Pr(R|R^*)Pr(R'^*|R^*)Pr(R^*)}{Pr(R)} \\
&= \sum_{R^* \in \{1,-1\}^{|R^*|}} Pr(R^*|R)Pr(R'^*) \\
&= Pr(R'^*)
\end{aligned}
$$

Notice the use of the assumption in the last step of the derivation. The result suggests that after receiving a set R of relationships from oracle calls, the posterior probabilities of all relationships r^*_{ij} that do not form any cycle of distinct objects with the relationships in R do not need to be computed - they remain the same as their priors. So, for example, after receiving the relationship r_{ij} from the first oracle call, only $Pr(r^*_{ij})$ needs to be updated, and which can be computed without using Equation 4. As compared to the exponential number of additions required to do the updates before we made the assumption, the reduction of computations is dramatic.

Now, suppose we want to compute the posterior probability of a relationship r^*_{ij}, which forms at least one cycle of distinct objects with some relationships in R. We want to partition R into two subsets R_{ij} and $\tilde{R}_{ij}$ such that there does not exist any cycle of distinct objects that involves both the edges in $R_{ij} \cup \{r^*_{ij}\}$ and the edges in $\tilde{R}_{ij}$ and no other edges. This partitioning can reduce the computational cost, because

$$
\begin{aligned}
Pr(r^*_{ij}, R^*_{ij}|R) &= Pr(r^*_{ij}, R^*_{ij}|R_{ij}, \tilde{R}_{ij}) \\
&= \alpha Pr(R_{ij}|r^*_{ij}, R^*_{ij}, \tilde{R}_{ij})Pr(r^*_{ij}, R^*_{ij}|\tilde{R}_{ij}) \\
&= \alpha Pr(R_{ij}|r^*_{ij}, R^*_{ij})Pr(r^*_{ij}, R^*_{ij}) \\
&= \alpha Pr(R_{ij}, r^*_{ij}, R^*_{ij}) \\
&= \alpha' Pr(r^*_{ij}, R^*_{ij}|R_{ij}) \qquad (5)
\end{aligned}
$$

This means if we can partition R into a number of subsets, so that each subset is independent of the others based on the restricted independence assumption, then we can compute the posterior of each subset separately. The smaller are the subsets, the fewer variables are involved in each subset, and the more efficient are the computations of the posterior probabilities.

Given a real relationship r^*_{ij} and a set of observed relationships R, we can find the minimal set R_{ij} that satisfies the above requirements by the following method. We ignore the relationship types, and treat R a collection of edges. Together with the edge (x_i, x_j), it becomes an undirected graph. We initialize R_{ij} as an empty set, and add all the relationships in R between x_i and x_j to it. Starting from x_i, we perform a breadth-first search. If a path reaches x_j, we take all the edges along the path, and add to R_{ij} all relationships in R whose object pair is an edge of the path. If a path reaches an object that has no unvisited edges, or an object that is already on the path, we stop extending the path. Then $\tilde{R}_{ij}$ is defined as $R - R_{ij}$.

We claim that after the search, $R_{ij} \cup \{r^*_{ij}\}$ and $\tilde{R}^*_{ij}$ satisfy the requirements of the assumption, so that they are independent. Also, R_{ij} is minimal, i.e., we cannot remove any relationship from it so that the requirements are still satisfied. The second part is trivial because each relationship in R_{ij} has its object pair participating in at least one cycle of distinct objects that involve (x_i, x_j) (we will call this kind of cycles an r^*_{ij}-cycle). So if we remove any relationship from R_{ij}, the cycles that this relationship participate would violate the requirements.

The first part can be proved by contradiction. Suppose there is a cycle of distinct objects that violates the requirements. This means the cycle must contain one or more edges that participates in an r^*_{ij}-cycle. If the cycle that causes the requirements violation involves r^*_{ij} or its complement, all the members in the cycle should be in R_{ij}, which is a contradiction. Otherwise, suppose some of the edges in the cycle participate in an r^*_{ij}-cycle. Starting from x_i, we follow the r^*_{ij}-cycle in the direction away from x_j, until we reach the first object that is also in the cycle that causes the requirements violation. Suppose the object is x'_i. We do the same thing, but this time we start from x_j, and follow the r^*_{ij}-cycle in the direction away form x_i. Suppose the object that we get is x'_j. Now, the cycle that causes the requirements violation must have a path from x_i and x_j that involves at least one edge whose object pair has no relationships in R_{ij}. Such edges should have been detected by the breadth-first search, and their relationships should be in R_{ij}, which is a contradiction.

We have discussed ways to determine which posterior probabilities need to be updated, and what variables need to be involved in the updates. Now we want to analyze the overall computational cost required if we want every posterior probability to be different from its prior. This means all posterior probabilities are supported by some observed relationships.

If we use the algorithm that involves AR and ER calls presented in Section 3, then if the oracles do not make errors, we can learn the whole relationship graph by receiving $|X| - 1$ relationships from oracle calls. If we treat the received relationships as edges, then the $|X| - 1$ relationships connect all $|X|$ objects. This means if we pick any pair of objects x_i, x_j, either we have received a relationship between them from the oracle calls, or the edge (x_i, x_j) can form a cycle with the edges of the received relationships. In other words, every relationship has its posterior updated by some observed relationships.

There is one way to minimize the total computational cost. If we make the AR calls for (x_1, x_2), (x_1, x_3), ..., $(x_1, x_{|X|})$, then each object pair either has a relationship received from an oracle call, or it participates in only one cycle, and the cycle involves only three variables. For example, the edge pair (x_4, x_9) participates only in the cycle (x_1, x_4, x_9). Therefore we can use Equation 5 to compute the joint posterior probabilities $Pr(r^*_{14}, r^*_{19}, r^*_{49}|R) = Pr(r^*_{14}, r^*_{19}, r^*_{49}|r_{14}, r_{19})$ in constant time. The total time complexity of computing all posterior probabilities is thus $O(|X|^2)$. Again, this shows that the restricted independence assumption of real relationships reduces the time complexity dramatically form exponential to polynomial.

We now return to an important question: what are the necessary conditions for the restricted independence assumption to hold? It turns out that a necessary condition for the assumption to hold is that the prior distribution of all consistent relationship graphs is uniform:

Lemma 10 *The restricted independence assumption of the real relationships holds only if the prior distribution over all consistent relationship graphs is uniform.*

Proof. Let us consider Table 1. Instead of having the constant probability of $\frac{1}{8}$ for all eight rows, suppose the prior probability for the event in row i to occur is p_i, for $1 \leq i \leq 8$. If the restricted independence assumption holds,

$$\begin{aligned} & Pr(r^*_{12} = 1 | r^*_{23} = 1) = Pr(r^*_{12} = 1) \\ \Rightarrow \quad & \frac{p_1 + p_8}{p_1 + p_4 + p_5 + p_8} = \frac{p_1 + p_2 + p_7 + p_8}{p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8} \\ \Rightarrow \quad & (p_1 + p_8)(p_3 + p_6) = (p_4 + p_5)(p_2 + p_7) \end{aligned}$$

Similarly,

$$\begin{aligned} & Pr(r^*_{13} = 1 | r^*_{23} = 1) = Pr(r^*_{13} = 1) \\ \Rightarrow \quad & \frac{p_1 + p_8}{p_1 + p_4 + p_5 + p_8} = \frac{p_1 + p_3 + p_6 + p_8}{p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8} \\ \Rightarrow \quad & (p_1 + p_8)(p_2 + p_7) = (p_4 + p_5)(p_3 + p_6) \end{aligned}$$

Combining the two equations, we get $p_3 + p_6 = p_2 + p_7$ and $p_1 + p_8 = p_4 + p_5$. Repeating with the given condition being $r^*_{13} = 1$ instead of $r^*_{23} = 1$, we get $p_4 + p_5 = p_2 + p_7$ and $p_1 + p_8 = p_3 + p_6$. Therefore, $p_1 + p_8 = p_2 + p_7 = p_3 + p_6 = p_4 + p_5$. This means all four consistent relationship assignments have the same probability of $\frac{1}{4}$. □

The assumption is therefore a very strong one that usually does not hold in reality. It is thus an interesting future work to study some weaker assumptions that also guarantees that the posterior probabilities can be computed in polynomial time.

Finally, we briefly talk about the use of the learnt posterior probabilities. If we simply want to learn E, the edges in the relationship graph of the target concept, then we can directly use the probabilities to answer the relationships between pairs of objects probabilistically. If instead we want to learn the target concept c, then we can compare different membership assignments and look for the one that maximizes the posterior probability of the resulting relationship graph. Notice that in general the result may not be unique. For example, if $\epsilon < 0.5$, and we have observed a cannot-link between all three pairs of objects in x_1, x_2, x_3, then each object pair has a higher posterior probability for cannot-link than must-link, and all three consistent relationship assignments that assign two cannot-links and one must-link to the three pairs have the same maximum posterior probability.

Alternatively, if we have the real membership of one or more objects, one may estimate the concept with the maximum posterior probability by using a Markov random walk approach, similar to the one proposed by Szummer and Jaakkola [15]. The basic idea is to diffuse the memberships probabilistically, where the probability for an object to diffuse its label to another object is based on the posterior probability of the must-link between the two objects, and the probability for an object to diffuse the negation of its label to another object is based on the posterior probability of the cannot-link between the two objects.

10 Discussion and future work

This study is intended to serve as a first attempt to understand learning from instance-level relationships in a theoretical setting. There are many problems remain unsolved, which could be interesting topics for future studies.

Most of the discussions in this article were focused on the general case where we know nothing about the concept class to which the target concept belongs. As shown in Example 1, when we have some knowledge about the concept class, we may actually obtain some results very different from the general case. For example, we might be able to learn the target concept using a much smaller number of oracle calls. One way to utilize the knowledge about concept classes is

to represent it as instance-level relationships, and use a general algorithm initialized with these relationships to learn the target concept. However, this approach might be computationally inefficient, and some knowledge may not be able to be represented as instance-level relationships. Another way is to represent the knowledge as new inference rules for deriving memberships and instance-level relationships.

We discussed an algorithm for learning a set of non-disjoint concepts, but we did not give a thorough analysis of the algorithm, nor did we discuss whether it is possible to have better algorithms that use fewer oracle calls. In fact, the algorithm uses a large number of calls to the five special oracles, which may induce a high cost in reality. It is interesting to study the necessary and sufficient conditions for the complete set of relationship graphs to be learnable by using the five kinds of oracles only. Another question is whether we need to learn the complete lattice and obtain all the disjointness and exhaustiveness information between the different nodes in order to minimize the number of calls to the individual AR oracles of the k target concepts. In real situations it is also possible to have only a fixed set of relationships of the form that the five oracles return. In such situations, the goal would be to learn the relationship graphs using a minimum of AR calls, subject to the constraints imposed by the fixed set of relationships.

Real oracles are likely to be noisy, so it is important to design good algorithms that learn from noisy oracles. As discussed in Section 9, there is a tradeoff between the validity of the assumption and the computational efficiency. A reasonable assumption that can facilitate efficient computation of the posterior probabilities is called for. Also, when discussing noisy oracles, we assumed that oracle calls are independent of each other, so that there is a fixed probability for an oracle to make an error. It might be more reasonable to adopt some other models. For example, if an oracle makes an error on a relationship that involves an object, the probability for it to make errors on other relationships that involve the same object may be relatively higher. For instance, if we ask for the relationship between the same pair of objects twice, it is very likely that if an oracle makes an error in the first time, it would also make an error in the second time. Therefore we may need some other models that do not assume independence between different oracle calls. This would also suggest that we should compute posterior probabilities based on observed relationships that involve different objects (rather than all based on the same object x_1 as suggested in Section 9), to reduce the risk that the relationships are all erroneous due to the dependency between them.

In most of the analyses, we considered the worst-case scenarios. It seems also useful to study the average cases. For example, the scanning algorithm for learning multiple disjoint concepts is not better than learning each concept individually in the worst case, but on average it is likely to perform much better because it could potentially eliminate many oracle calls that cannot not bring any new information. By considering average cases, there may be learning algorithms that are better than the ones proposed in this article.

Finally, besides must-links, cannot-links, and the five types of relationships defined for multiple non-disjoint concepts, it is also interesting to explore other types of relationships as well as other kinds of information that could be used in learning.

11 Summary and conclusion

In this article, we have studied various problems of learning from instance-level constraints. We have focused the study on two types of instance-level constraints: must-links that state that two objects are both in a target concept or are both not in the concept, and cannot-links that state that exactly one object of a pair is in the target concept. We have assumed that relationships can be obtained from some oracles, either active ones that can answer relationships between a specific pair of objects or whether the specified concept is exactly the target concept or its complement, or the passive one that returns the relationships of random object pairs.

We have discussed that a lot of research efforts on machine learning have been made on learning from labeled examples, and that actually there are other types of information that can be utilized, such as instance-level relationships. We have suggested example applications of learning from instance-level constraints in bioinformatics.

We have discussed a general learning method that learns a target concept completely using an arbitrary sequence of AR, ER and PR oracles. We have shown that if each oracle call returns some new information, this algorithm requires exactly $|X| - 1$ oracle calls, and runs in linear time and space, where X is the set of objects over which the target concept c defines. We have also discussed ways to make AR and ER calls, such that each call is guaranteed to return some new information.

On the topic of active learning, we have shown how AR and AM oracles can simulate each other, as well as how ER and EM oracles can simulate each other, with the help of some AR calls when ER is simulating EM. Based on the simulations, we have shown that each learning algorithm that makes use of AM and EM can be simulated by using the same order of AR and ER calls.

We have introduced the relationship concept class of a concept class, which treats object pairs as higher-order objects, and the relationship graph (the graph (X, E) where E is the set of all must-linked object pairs) as a higher-order concept. We have proved the tight lower and upper bounds of the VC-dimension of a relationship concept class in terms of the VC-dimension of the corresponding concept class and the sizes of the largest and smallest concepts in it.

For passive learning, we have given the necessary and sufficient conditions for a target concept to be completely learnt by using PR calls. Also, using the lower and upper bounds of the VC-dimension of a relationship class, we have shown how some previous results on PAC-learning can be applied to learning from instance-level constraints.

We have also discussed mixed learning that involves both active and passive oracles. We have based our discussions on several reward functions, and have discussed how the best actions can be determined theoretically, as well as some practical issues.

Since there are situations that we want to learn multiple concepts at the same time, we have studied learning in a multi-concept environment. For k non-empty disjoint concepts, we have described an algorithm that learns the relationship graphs of the concepts completely using $(k-1)(|X| - \frac{k}{2})$ oracle calls. We have proved that this is the minimum number of oracle calls required in the worst case, which suggests that in the worst case there is no real benefit to learn the disjoint concepts all at the same time than learning them one by one, if the number of concepts is much smaller than the number of objects. For the non-disjoint case, we have suggested five different types of oracles that provide different kinds of information about the relationships between the two sets of concepts that two objects belong to. We have discussed how the oracles can be used to partition objects into groups such that all objects in

a group all belong to exactly the same set of concepts, and to learn the subset, disjointness and exhaustiveness information between the different groups. We have also discussed how such information can be used to reduce the number of, or even completely avoid making, calls to the individual AR oracles of the concepts.

Finally, we have discussed issues in learning from noisy oracles. We have assumed that each oracle has a fixed error rate of returning an incorrect relationship between a pair of objects. We have setup a Bayesian probabilistic framework for inputting prior probabilities of the relationships between pairs of objects and computing posterior probabilities after observing some relationships from oracles. We have shown that in the general case, computing the posterior probabilities requires an exponential number of summations. We have suggested a restricted independence assumption of the real relationships, and have shown that it reduces the time complexity to quadratic. However, we have also shown that a necessary requirement for the assumption to hold is a uniform prior over all consistent relationship graphs, which makes the assumption too strong to be valid in real situations.

To conclude, we have shown that instance-level relationships are useful in learning. They can be potentially utilized to reduce the total learning cost, and improve the accuracy of the learnt concept when there is a limited supply of labeled examples. It is hoped that the current study would simulate more ideas on semi-supervised learning in general.

12 Acknowledgement

The author would like to thank Prof. Dana Angluin, Samuel Daitch, Viksit Gaur, Chris Malizia and Tim McDermott for their invaluable suggestions and comments.

References

[1] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs genomic sequence alignment. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[4] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 19–26, 2002.

[5] S. Basu, A. Banerjee, and R. J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining*, pages 333–344, 2004.

[6] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68, 2004.

[7] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 81–88, 2004.

[8] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Eleventh Annual Conference on Learning Theory (COLT)*, pages 92–100, 1998.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[10] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. Technical Report TR2003-1892, Cornell University, 2003.

[11] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Engineering Systems through Artificial Neural Networks*, volume 9, pages 809–814, 1999.

[12] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[13] D. Klein, S. D. Kamvar, and C. D. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 307–314, 2002.

[14] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[15] M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing Systems 14*, pages 945–952, 2002.

[16] L. Talavera and J. Bejar. Integrating declarative knowledge in hierarchical clustering tasks. In *International Symposium on Intelligent Data Analysis*, pages 211–222, 1999.

[17] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[18] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110, 2000.

[19] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, 2001.

[20] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512, 2003.

[21] K. Y. Yip, D. W. Cheung, and M. K. Ng. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering. In *IEEE International Conference on Data Engineering (ICDE)*, pages 329–340, 2005.