JMLR: Workshop and Conference Proceedings vol 40:1–20, 2015

S^2 : An Efficient Graph Based Active Learning Algorithm with Application to Nonparametric Classification

Gautam Dasarathy GAUTAMD@CS.CMU.EDU
Carnegie Mellon University

Robert Nowak RDNOWAK@WISC.EDU
Xiaojin Zhu JERRYZHU@CS.WISC.EDU
University of Wisconsin - Madison

Abstract

This paper investigates the problem of active learning for binary label prediction on a graph. We introduce a simple and label-efficient algorithm called S^2 for this task. At each step, S^2 selects the vertex to be labeled based on the structure of the graph and all previously gathered labels. Specifically, S^2 queries for the label of the vertex that bisects the *shortest shortest* path between any pair of oppositely labeled vertices. We present a theoretical estimate of the number of queries S^2 needs in terms of a novel parametrization of the complexity of binary functions on graphs. We also present experimental results demonstrating the performance of S^2 on both real and synthetic data. While other graph-based active learning algorithms have shown promise in practice, our algorithm is the first with both good performance and theoretical guarantees. Finally, we demonstrate the implications of the S^2 algorithm to the theory of nonparametric active learning. In particular, we show that S^2 achieves near minimax optimal excess risk for an important class of nonparametric classification problems.

Keywords: active learning on graphs, query complexity of finding a cut, nonparametric classification

1. Introduction

This paper studies the problem of binary label prediction on a graph. We suppose that we are given a graph over a set of vertices, where each vertex is associated with an initially unknown binary label. For instance, the vertices could represent objects from two classes, and the graph could represent the structure of similarity of these objects. The unknown labels then indicate the class that each object belongs to. The goal of the general problem of binary label prediction on a graph is to predict the label of all the vertices given (possibly noisy) labels for a subset of the vertices. Obtaining this initial set of labels could be costly as it may involve consulting human experts or expensive experiments. It is therefore of considerable interest to minimize the number of vertices whose labels need to be revealed before the algorithm can predict the remaining labels accurately. In this paper, we are especially interested in designing an *active* algorithm for addressing this problem, that is, an algorithm that sequentially and automatically selects the vertices to be labeled based on both the structure of the graph and the previously gathered labels. We will now highlight the main contributions of the paper:

- **A new active learning algorithm which we call S^2.** In essence, S^2 , which stands for *shortest shortest path* operates as follows. Given a graph and a subset of vertices that have been

labeled, S^2 picks the mid-point of the path of least length among all the shortest paths connecting oppositely labeled vertices. As we will demonstrate in Sec 4, S^2 automatically adapts itself to a natural notion of the complexity of the cut-set of the labeled graph.

- **A novel complexity measure**. While prior work on graph label prediction has focused only on the cut-size (i.e., the number of edges that connect oppositely labeled nodes) of the labeling, our refined complexity measure (cf. Section 4.1) quantifies the difficulty of learning the cut-set. Roughly speaking, it measures how clustered the cut-set is; the more clustered the cut-set, the easier it is to find. This is analogous to the fact that the difficulty of classification problems in standard settings depends both on the *size* and the *complexity* of the Bayes decision boundary.

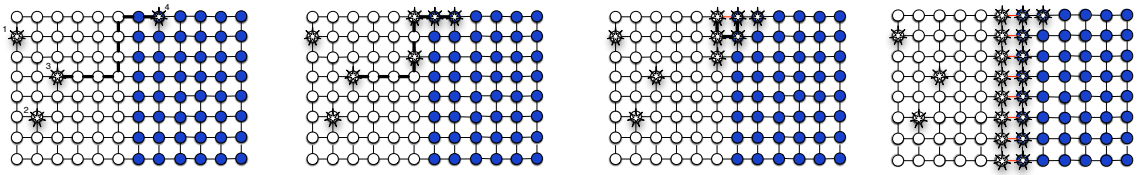
- **A practical algorithm for non-parametric active learning.** Significant progress has been made in terms of characterizing the theoretical advantages of active learning in nonparametric settings (e.g., Castro and Nowak (2008); Wang (2011); Minsker (2012); Hanneke (2011); Koltchinskii (2010)), but most methods are not easy to apply in practice. On the other hand, the algorithm proposed in Zhu et al. (2003b), for example, offers a flexible approach to non-parametric active learning that appears to provide good results in practice. It however does not come with theoretical performance guarantees. A contribution of our paper is to fill the gap between the practical method of Zhu et al. (2003b) and the theoretical work above. We show that S^2 achieves the minimax rate of convergence for classification problems with decision boundaries in the so-called *box-counting* class (see Castro and Nowak (2008)). To the best of our knowledge this is the first practical algorithm that is minimax-optimal (up to logarithmic factors) for this class of problems.

2. Preliminaries and Problem Setup

We will write $G = (V, E)$ to denote an undirected graph with vertex set V and edge set E. Let $f : V \to \{-1, +1\}$ be a binary function on the vertex set. We call $f(v)$ the *label* of $v \in V$. We will further suppose that we only have access to the labeling function f through a (potentially) noisy oracle. That is, fixing $\gamma \in (0, 0.5]$, we will assume that for each vertex $v \in V$, the oracle returns a random variable $\hat{f}(v)$ such that $\hat{f}(v)$ equals $-f(v)$ with probability γ and equals $f(v)$ with probability $1 - \gamma$, independently of any other query submitted to this oracle. We will refer to such an oracle as a $\gamma-$*noisy* oracle. The goal then is to design an algorithm which sequentially selects a multiset[1] of vertices L and uses the labels $\left\{\hat{f}(v), v \in L\right\}$ to accurately learn the true labeling function f. Since the algorithm we design will assume nothing about the labeling, it will be equipped to handle even an adversarial labeling of the graph. Towards this end, if we let C denote the *cut-set* of the labeled graph, i.e., $C \triangleq \{\{x, y\} \in E : f(x) \neq f(y)\}$ and let ∂C denote the *boundary* of the cut-set, i.e., $\partial C = \{x \in V : \exists e \in C \text{ with } x \in e\}$, then our goal will actually be to identify ∂C.

Of course, it is of interest to make L as small as possible. Given $\epsilon \in (0, 1)$, the number of vertices that an algorithm requires the oracle to label, so that with probability at least $1 - \epsilon$ the algorithm correctly learns f will be referred to as its $\epsilon-$*query complexity*. We will now show that

1. A multiset is a set that may contain repeated elements.



(*a*) Random sampling ends. One shortest shortest path is shown with thickened edges.

(*b*) This path bisected and S^2 finds one cut edge.

(*c*) Next shortest-shortest path is bisected.

(*d*) This continues till S^2 unzips through the cut boundary.

Figure 1: A sample run of the S^2 algorithm on a grid graph. The shaded and unshaded vertices represent two different classes (say $+1$ and -1 respectively). See text for explanation.

given an algorithm that performs well with a noiseless oracle, one can design an algorithm that performs well on a $\gamma-$noisy oracle.

Proposition 1 *Suppose $\mathcal{A}$ is an algorithm that has access to f through a noiseless oracle, and suppose that it has a $\epsilon-$query complexity q, then for each $\gamma \in (0, 0.5)$, there exists an algorithm $\tilde{\mathcal{A}}$ which, using a $\gamma-$noisy oracle achieves a $2\epsilon-$query complexity given by $q \times \left\lceil \frac{1}{2(0.5-\gamma)^2} \log\left(\frac{n}{\epsilon}\right) \right\rceil$.*

The main idea behind this proposition is that one can build a noise-tolerant version of $\mathcal{A}$ by repeating each query that $\mathcal{A}$ requires many times, and using the majority vote. The proof of Proposition 1 is then a straightforward application of Chernoff bounds and we defer it to Appendix A.

Therefore, to keep our presentation simple, we will assume in the sequel that our oracle is noiseless. A more nuanced treatment of noisy oracles is an interesting avenue for future work.

It should also be noted here that the results in this paper can be extended to the multi-class setting, where $f : V \to \{1, 2, \ldots, k\}$ by the standard "one-vs-rest" heuristic (see e.g., Bishop et al. (2006)). However, a thorough investigation of the multiclass classification on graphs is an interesting avenue for future work.

3. The S^2 algorithm

The name S^2 signifies the fact that the algorithm bisects the *shortest shortest-path* connecting oppositely labeled vertices in the graph. As we will see, this allows the algorithm to automatically take advantage of the clusteredness of the cut set. Therefore, S^2 "*unzips*" through a tightly clustered cut set and locates it rapidly (see Figure 1).

The algorithm works by first constructing a label set $L \subset V$ sequentially and adaptively and then using the labels in L to predict the labels of the vertices in $V \setminus L$. It accepts as input a natural number BUDGET, which is the query budget that we are given to complete the learning task. In our theoretical analysis, we will show how big this budget needs to be in order to perform well on a wide variety of problems. Of course this budget specification merely reflects the completely agnostic nature of the algorithm; we subsequently show (see Section 3.1) how one might factor in expert knowledge about the problem to create more useful stopping criteria.

Algorithm 1: S^2 : Shortest Shortest Path

Input Graph $G = (V, E)$, BUDGET $\leq n$

1: $L \leftarrow \emptyset$
2: **while** 1 **do**
3: $x \leftarrow$ Randomly chosen unlabeled vertex
4: **do**
5: Add $(x, f(x))$ to L
6: Remove from G all edges whose two ends have different labels.
7: **if** $|L|$ = BUDGET **then**
8: **Return** LABELCOMPLETION(G, L)
9: **end if**
10: **while** $x \leftarrow$ MSSP(G, L) exists
11: **end while**

Sub-routine 2: MSSP

Input Graph $G = (V, E)$, $L \subseteq V$

1: **for** each $v_i, v_j \in L$ such that $f(v_i) \neq f(v_j)$ **do**
2: $P_{ij} \leftarrow$ shortest path between v_i and v_j in G
3: $\ell_{ij} \leftarrow$ length of P_{ij} (∞ if no path exists)
4: **end for**
5: $(i^*, j^*) \leftarrow \arg\min_{v_i, v_j \in L: f(v_i) \neq f(v_j)} \ell_{ij}$
6: **if** (i^*, j^*) exists **then**
7: **Return** mid-point of $P_{i^*j^*}$ (break ties arbitrarily).
8: **else**
9: **Return** $\emptyset$
10: **end if**

S^2 (Algorithm 1) accepts a graph G and a natural number BUDGET. Step 3 performs a random query. In step 6, the obvious cut edges are identified and removed from the graph. This ensures that once a cut edge is found, we do not expend more queries on it. In step 7, the algorithm ensures that it has enough budget left to proceed. If not, it stops querying for labels and completes the labeling using the subroutine LABELCOMPLETION. In step 10, the mid-point of the shortest shortest path is found using the MSSP subroutine (Subroutine 2) and the algorithm proceeds till there are no more mid-points to find. Then, the algorithm performs another random query and the above procedure is repeated. As discussed in Proposition 1, if in Step 5, we compute $f(x)$ as the majority label among $\mathcal{O}(\log(n/\epsilon))$ repeated queries about the label of x, then we get a *noise tolerant version* of S^2 .

We will now describe the sub-routines used by S^2 . Each of these accepts a graph G and a set L of vertices. LABELCOMPLETION(G, L) is any off-the-shelf procedure that can complete the labeling of a graph from a subset of labels. This could, for instance, be the graph min-cut procedure Blum and Chawla (2001) or harmonic label propagation Zhu et al. (2003a). We only include this sub-routine for the sake of completeness with respect to any given budget. Our theoretical results do not depend on this subroutine. MSSP(G, L) is Subroutine 2 and finds the midpoint on the shortest among all the shortest-paths that connect oppositely labeled vertices in L. If none exist, it returns $\emptyset$.

The main idea underlying the S^2 algorithm is the fact that learning the labeling function f amounts to locating all the *cut-edges* in the labeled graph. Conceptually, the algorithm can be thought of as operating in two phases: random sampling and aggressive search. In the random sampling phase, the algorithm queries randomly chosen unlabeled vertices until it finds two vertices with opposite labels. Then our algorithm enters the aggressive search phase. It picks the shortest path between these two points and bisects it. What our algorithm does next sets it apart from prior work such as Afshani et al. (2007). It does not run each binary search to the end, but merely

keeps bisecting the shortest among all the shortest paths that connect oppositely labeled vertices observed so far. This endows the algorithm with the ability to "unzip" cut boundaries. Consider a sample run shown in Figure 1. The random sampling phase first picks a set of random vertices till an oppositely labeled pair is observed as in Figure 1(a). The shortest shortest path connecting oppositely labeled nodes is shown here as a thick sequence of edges. Figure 1(b) shows that S^2 now bisects shortest shortest paths and subsequently finds a cut-edge. The bisection of the next two shortest shortest paths is shows in Figure 1(c) and we see the boundary unzipping feature of the algorithm emerges. Figure 1(d) finally shows the situation after the completion of the algorithm. Notice that an extremely small number of queries are used before S^2 completely discovers the cut boundary.

3.1. Stopping Criterion

Notice S^2 stops only if the budget is exhausted. This stopping criterion was chosen for two main reasons. Firstly, this keeps the presentation simple and reflects the fact that S^2 *assumes nothing* about the underlying problem. In practice, extra information about the problem can be easily incorporated. For instance, suppose one knows a bound on the cut-set or on the size of similarly labeled connected components, then such criteria can be used in Step 7. Similarly, one can hold out a random subset of observed labels and stop upon achieving low prediction error on this subset. Secondly, in our theoretical analysis, we show that as long as the budget is large enough, then S^2 recovers the cut boundary exactly. The larger the budget is, the more complex the graphs and labelings S^2 can handle. Therefore, our result can be interpreted as a quantification of the complexity of a natural class of graphs. In that sense S^2 is not only an algorithm but a tool for exposing the theoretical challenges of label prediction on a graph.

4. Analysis

Let C and ∂C be as defined in Section 2. As we observed earlier, the problem of learning the labeling function f is equivalent to the problem of locating all the cut edges in the graph. Clearly, if f could be arbitrary, the task of learning f given its value on a subset of its domain is ill-posed. However, we can make this problem interesting by constraining the class of functions that f is allowed to be in. Towards this end, in the next section we will discuss the complexity of a labeling function with respect to the underlying graph.

4.1. Complexity of the Labeling Function with respect to the Graph

We will begin by making a simple observation. Given a graph G and a labeling function f, notice that f partitions the vertices of the graph into a collection of connected components with identically labeled vertices. We will denote these connected components as $V_1, V_2, \ldots, V_k$, where k represents the number of connected components.

Then, the above partitioning of the vertex set induces a natural partitioning of the cut set $C = \bigsqcup_{1 \le r < s \le k} C_{rs}$, where $C_{rs} = \{x, y \in C : x \in V_r, y \in V_s\}$. That is, C_{rs} contains the cut edges whose boundary points are in the components V_r and V_s[2]. We denote the number of non-empty subsets C_{rs} by m, and we call each non-empty C_{rs} a *cut component*. See Figure 2(a) for an

2. C_{rs} could of course be empty for certain pairs r, s.

illustration. It shows a graph, its corresponding labeling (denoted by darker and lighter colored vertices) and the cut set (thickened lines). It also shows the induced vertex components V_1, V_2, V_3 and the corresponding cut components C_{12} and C_{23}. We will especially be interested in how clustered a particular cut component is. For instance, compare the cut component C_{12} between Figures 2(a) and 2(b). Intuitively, it is clear that the former is more clustered than the latter. As one might expect, we show that it is easier to locate well-clustered cut components.

We will now introduce three parameters which, we argue, naturally govern the complexity of a particular problem instance. Our main results will be expressed in terms of these parameters.

1. Boundary Size. The first and the most natural measure of complexity we consider is the size of the boundary of the cut set $|\partial C|$. It is not hard to see that a problem instance is hard if the boundary size induced by the labeling is large. In fact, $|\partial C|$ is trivially a lower bound on the total number of queries needed to learn the location of C. Conversely, if it is the case that well-connected vertices predominantly have the same label, then $|\partial C|$ will most likely be small. Theorem 3 will show that the number of queries needed by the S^2 algorithm scales approximately linearly with $|\partial C|$.

2. Balancedness. The next notion of label complexity we consider is the *balancedness* of the labeling. As we discussed above, the labeling function induces a partition on the vertex set $V_1, V_2, \ldots, V_k$. We will define the *balancedness of f with respect to G* as $\beta \triangleq \min_{1 \leq i \leq k} \frac{|V_i|}{n}$.

It is not hard to see why balancedness is a natural way of quantifying problem complexity. If there is a very small component, then without prior knowledge, it will be unlikely that we see labeled vertices from this component. We would then have no reason to assign labels to its vertices that are different from its neighbors. Therefore, as we might expect, the larger the value of β, the easier it is to find a particular labeling (see Lemma 4 for more on this).

3. Clustering of Cut Components. We finally introduce a notion of complexity of the labeling function which, to the best of our knowledge, is novel. As we show, this complexity parameter is key to developing an efficient active learning algorithm for general graphs.

Let $d_{\text{sp}}^G : V \times V \to \mathbb{N} \cup \{0, \infty\}$ denote the shortest path metric with respect to the graph G, i.e., $d_{\text{sp}}^G(x, y)$ is the the length of the shortest path connecting x and y in G with the convention that the distance is ∞ if x and y are disconnected. Using this, we will define a metric $\delta : C \times C \to \mathbb{N} \cup \{0, \infty\}$ on the cut-edges as follows. Let $e_1 = \{x_1, y_1\}, e_2 = \{x_2, y_2\} \in C$ be such that $f(x_1) = f(x_2) = +1$ and $f(y_1) = f(y_2) = -1$. Then $\delta(e_1, e_2) = d_{\text{sp}}^{G-C}(x_1, x_2) + d_{\text{sp}}^{G-C}(y_1, y_2) + 1$, where $G - C$ is the graph G with all the cut-edges removed. Notice that δ is a metric on C and that $\delta(e_1, e_2) < \infty$ if and only if e_1 and e_2 lie in the same cut-component.

Imagine a scenario where a cut-component C_{rs} is such that each pair $e, e' \in C_{rs}$ satisfy $\delta(e, e') \leq \kappa$. Now, suppose that the end points of one of the cut-edges $e_1 \in C_{rs}$ has been discovered. By assumption, each remaining cut-edge in C_{rs} lies in a path of length at most κ from a pair of oppositely labeled vertices (i.e., the end points of e_1). Therefore, if one were to do a bisection search on each path connecting these oppositely labeled vertices, one can find all the cut-edges using no more that $|C_{rs}| \log \kappa$ queries. If κ is small, this quantity could be significantly smaller than n (which is what an exhaustive querying algorithm would do). The reason for the drastically reduced query complexity is the fact that the cut-edges in C_{rs} are *tightly clustered*. A generalization of this observation gives rise to our third notion of complexity.

Let $H_r = (C, \mathcal{E})$ be a "meta graph" whose vertices are the cut-edges of G and $\{e, e'\} \in \mathcal{E}$ iff $\delta(e, e') \leq r$, i.e., H_r is the r-nearest neighbor graph of the cut-edges, where distances are defined

according to δ. From the definition of δ, it is clear that for any $r \in \mathbb{N}$, H_r has at least m connected components.

Definition 2 *A cut-set C is said to be $\kappa-$***clustered** *if H_κ has exactly m connected components. These connected components correspond to the cut-components, and we will say that each individual cut-component is also κ-clustered.*

Turning back to Figure 2, observe that the cut component C_{12} is κ-clustered for any $\kappa \geq 5$ in Figure 2(a) and $\kappa-$clustered for any $\kappa \geq 12$ in Figure 2(b). The figure also shows a length 5 (resp. 12) path in Fig 2(a) (resp. Fig 2(b)) that defines the clusteredness.

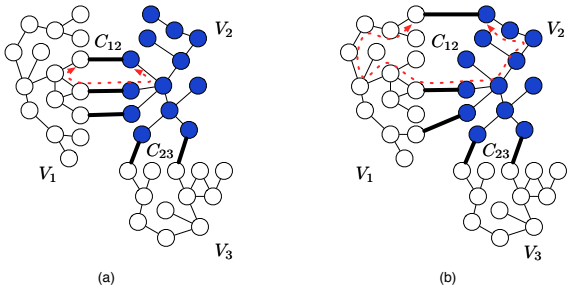


Figure 2: Two graphs with the same number of cut-edges (thickened edges) but the cut component C_{12} in (a) is more "clustered" than its counterpart in (b). C_{12} is 5-clustered in (a) and 12-clustered in (b). The corresponding length κ paths are shows with a (red) dotted line.

4.2. The query complexity of S^2

In this section, we will prove the following theorem.

Theorem 3 *Suppose that a graph $G = (V, E)$ and a binary function f are such that the induced cut set C has m cut components that are each $\kappa-$clustered. Then for any $\epsilon > 0$, S^2 will recover C with probability at least $1 - \epsilon$ if the* BUDGET *is at least*

$$\frac{\log(1/(\beta\epsilon))}{\log\left(1/(1-\beta)\right)} + m\left(\lceil \log_2 n \rceil - \lceil \log_2 \kappa \rceil\right) + |\partial C|\left(\lceil \log_2 \kappa \rceil + 1\right) \tag{1}$$

Before we prove the theorem, we will make some remarks:

1. As observed in Afshani et al. (2007), using $\mathcal{O}(|\partial C|)$ queries, we can create a greedy cover of the graph and reduce the length of the longest binary search from n to $\mathcal{O}\left(\frac{n}{|\partial C|}\right)$. With this simple modification to S^2 , the $\log_2 n$ in (1) can be replaced by $\log(n/\,|\partial C|)$.

2. Suppose that G is a $\sqrt{n} \times \sqrt{n}$ grid graph where one half of this graph, i.e., a $\sqrt{n} \times \sqrt{n}/2$ rectangular grid, is labeled $+1$ and the other half -1. In this case, since $m = 1, |\partial C| = \mathcal{O}(\sqrt{n})$ and $\kappa = 3$, Theorem 3 says that S^2 needs to query at most $\mathcal{O}(\sqrt{n} + \log n)$ vertices. This is much smaller than the bound of $\mathcal{O}\left(\sqrt{n}\log n\right)$ which can be obtained using the results of Afshani et al. (2007).

In fact, when $\kappa < n/|\partial C|$, it is clear that for $m \geq 1$, $m\log(n/|\partial C|) + (|\partial C| - m)\log\kappa < |\partial C|\log(n/|\partial C|)$, i.e., our bounds are strictly better than those in Afshani et al. (2007) [3].

3. the first term in (1) is due to random sampling and is present in both results. We have also ignored integer effects in making this observation.

3. As discussed in Section 2, the number of queries the (i.i.d) noise tolerant version of S^2 submits can be bounded by the above quantity times $\left\lceil \frac{1}{2\gamma^2} \log\left(\frac{n}{\epsilon}\right) \right\rceil$ and is guaranteed to recover C exactly with probability at least $1 - 2\epsilon$.

4. The first term in (1) is due to random sampling phase which can be quite large if β is very small. Intuitively, if V_i is a very small connected component, then one might have to query almost all vertices before one can even obtain a label from V_i. Therefore, the "balanced" situation, where β is a constant independent of n, is ideal here. These considerations come up in Afshani et al. (2007) as well, and in fact their algorithm needs a priori knowledge of β. Such balancedness assumptions arise in various other lines of work as well (e.g., Eriksson et al. (2011); Balakrishnan et al. (2011)). Finally, we remark that if one is willing to ignore errors made on small "outlier" components, then β can be redefined in terms of only the sufficiently large V_i's. This allows us to readily generalize our results to the case where one can approximately recover the labeling of the graph.

5. As we argue in Appendix D, S^2 is near optimal with respect to the complexity parametrization introduced in this paper. That is, we show that there exists a family of graphs such that *no algorithm* has significantly lower query complexity than S^2 . It will be interesting to investigate the "instance-optimality" property of S^2 , where we fix a graph and then ask if S^2 is near optimal in discovering labelings on this graph. We take a step in this direction in Appendix D.2 and show that for the 2-dimensional grid graph, the number of queries made by S^2 is near optimal.

Proof of Theorem 3 We will begin the proof with a definition and an observation. Recall (from Section 4.1) that the labeling function partitions the vertex set into similarly labeled components $V_1, \ldots, V_k$. Suppose that $W \subset V$ is such that for each $i \in \{1, 2, \ldots, k\}$, $|W \cap V_i| \geq 1$. Then, it follows that for each $e \in C$, there exists a pair of vertices v, v' in W such that $f(v) \neq f(v')$ and e lies on a path joining v and v'. We call such a set a *witness* to the cutset C. Since ∂C is a witness to the cut-set C, it is clearly necessary for any algorithm to know the labels of a witness to the cut set C in order to learn the cut set.

Now, as described in Section 3, the operation of S^2 consists of two phases – (a) the random sampling phase and (b) the aggressive search phase. Observe that if S^2 knows the labels of a witness to the cut component C_{rs}, then, until it discovers the entire cut-component, S^2 remains in the aggressive search phase. Therefore, the goal of the random sampling phase is to find a witness to the cut set. This implies that we can bound from above the number of random queries S^2 needs before it can locate a witness set, and this is done in the following lemma.

Lemma 4 *Consider a graph $\mathcal{G} = (V, E)$ and a labeling function f with balancedness β. For all $\alpha > 0$, a subset L chosen uniformly at random is a witness to the cut-set with probability at least $1 - \alpha$, as long as $|L| \geq \frac{\log(1/(\beta\alpha))}{\log(1/(1-\beta))}$.*

We refer the reader to Appendix B for the proof which is a straightforward combinatorial argument.

Of course, since the random sampling phase and the aggressive search phase are interleaved, S^2 might end up needing far fewer random queries before a witness W to the cut-set has been identified.

Now, we will turn our attention to the aggressive search phase. The algorithm enters the aggressive search phase as soon there are oppositely labeled vertices that are connected (recall that the algorithm removes any cut-edge that it finds). Let $\ell_{S^2}(G, L, f)$ be the length of the shortest among all shortest paths connecting oppositely labeled vertices in L, and we will suppress this quantity's dependence on G, L, f when the context is clear. After each step of the aggressive search phase, the shortest shortest paths that connect oppositely labeled vertices gets bisected. Suppose that, during

this phase, the current shortest path length is $\ell_{S^2} = \ell \in \mathbb{N}$. Then depending on its parity, after one step ℓ_{S^2} gets updated to $\frac{\ell+1}{2}$ (if ℓ is odd) or $\frac{\ell}{2}$ (if ℓ is even). Therefore, we have the following result .

Claim 1 *If $\ell_{S^2} = \ell$, after no more than $r = \lceil \log_2 \ell \rceil + 1$ aggressive steps, a cut-edge is found.*

Proof First observe that after r steps of the aggressive search phase, the length of the shortest shortest path is no more than $\frac{\ell+2r-1}{2^r}$. Next, observe that if $r \geq 4$, then $2^{r-1} \leq 2^r - 2r + 1$.

The proof of this claim then follows from the following observation. Let us first suppose that $\ell \geq 8$, then setting $r = \lceil \log_2 \ell \rceil + 1$, we have that $\ell \leq 2^{r-1} \leq 2^r - 2r + 1$, which of course implies that $\frac{\ell+2r-1}{2^r} \leq 1$. Therefore, after at most r steps, the current shortest shortest path length drops to below 1, i.e., a cut-edge will be found. For the case when $\ell < 8$, one can exhaustively check that this statement is true. ■

It is instructive to observe two things here: (a) even though ℓ_{S^2} gets (nearly) halved after every such aggressive step, it might not correspond to the length of a single path through the span of the above r steps, and (b) at the end of r aggressive steps as above, at least one new boundary vertex is found, therefore, the algorithm might end up uncovering multiple cut-edges after r steps.

To bound the number of active queries needed, let us split up the aggressive search phase of S^2 into "runs", where each run ends when a new boundary vertex has been discovered, and commences either after the random sampling phase exposes a new path that connects oppositely labeled vertices or when the previous run ends in a new boundary vertex being discovered. Let R be the total number of runs. Notice that $R \leq |\partial C|$. For each $i \in R$, let G_i and L_i denote the graph and the label set at the start of run i. Therefore, by Claim 1, the total number of active learning queries needed can be upper bounded by $\sum_{i \in R} \{\lceil \log_2 (\ell_{S^2}(G_i, L_i, f))\rceil + 1\}$.

Now, observe that for each run in $i \in R$, it trivially holds that $\ell_{S^2}(G_i, L_i, f) \leq n$ [§]. Now suppose that one cut-edge is discovered in a cut-component C_{rs}. Since the graph on C_{rs} induced by H_κ is κ-connected by the assumption of the theorem, until all the cut-edges are discovered, there exists at least one undiscovered cut-edge in C_{rs} that is at most κ away (in the sense of δ) from one of the discovered cut-edges. Therefore, ℓ_{S^2} is no more than κ until all cuts in C_{rs} have been discovered. In other words, for $|C_{rs}| - 1$ runs in R, $\ell_{S^2} \leq \kappa$.

Reasoning similarly for each of the m cut components, we have that there are no more than m "long" runs (one per cut-component) and we will bound ℓ_{S^2} for these runs by n. From the argument above, after an edge from a cut-component has been found, ℓ_{S^2} is never more than κ till all the edges of the cut-component are discovered. Therefore, we have the following upper bound on the number of aggressive queries needed.

$$\begin{aligned}\sum_{i \in R} \{\lceil \log_2 (\ell_{S^2}(G_i, L_i, f))\rceil + 1\} &\leq |\partial C| + m\lceil \log_2 n \rceil + (|\partial C| - m)\lceil \log_2 \kappa \rceil \\ &= m\left(\lceil \log_2 n \rceil - \lceil \log_2 \kappa \rceil\right) + |\partial C|\left(\lceil \log_2 \kappa \rceil + 1\right)\end{aligned}$$

Putting this together with the upper bound on the number of random queries needed, we get the desired result. ■

§. As in the Remark 1 after the statement of the theorem, this can be improved to $\mathcal{O}\left(n/\left|\partial C\right|\right)$ using a greedy cover of the graph (Afshani et al., 2007).

5. S^2 for Nonparametric Learning

Significant progress has been made in terms of characterizing the theoretical advantages of active learning in nonparametric settings. For instance, minimax rates of convergence in active learning have been characterized under the so called *boundary fragment* assumption (Castro and Nowak, 2008; Wang, 2011; Minsker, 2012). This model requires the Bayes decision boundary to have a functional form. For example, if the feature space is $[0,1]^d$, then the boundary fragment model assumes that the Bayes decision boundary is defined by a curve of the form $x_d = f(x_1, \ldots, x_{d-1})$, for some (smooth) function f. While such assumptions have proved useful for theoretical analysis, they are unrealistic in practice. Nonparametric active learning has also been analyzed in terms of abstract concepts such as bracketing and covering numbers (Hanneke, 2011; Koltchinskii, 2010), but it can be difficult to apply these tools in practice as well. The algorithm proposed in Zhu et al. (2003b) offers a flexible approach to nonparametric active learning that appears to provide good results in practice, but comes with no theoretical performance guarantees. A contribution of our paper is to fill the gap between the practical method of Zhu et al. (2003b) and the theoretical work above. The S^2 algorithm can adapt to nonparametric decision boundaries without the unrealistic boundary fragment assumption required by Castro and Nowak (2008), for example. We show that S^2 achieves the minimax rate of convergence for classification problems with decision boundaries in the so-called *box-counting* class, which is far less restrictive than the boundary fragment model. To the best of our knowledge this is the first practical algorithm that is near minimax-optimal for this class of problems.

5.1. Box-Counting Class

Consider a binary classification problem on the feature space $[0,1]^d$, $d \geq 1$. The box-counting class of decision boundaries generalizes the set of boundary fragments with Lipschitz regularity to sets with arbitrary orientations, piecewise smoothness, and multiple connected components. Thus, it is a more realistic assumption than boundary fragments for classification; see Scott and Nowak (2006) for more details. Let w be an integer and let P_w denote the regular partition of $[0,1]^d$ into hypercubes of side length $1/w$. Every classification rule can be specified by a set $B \subset [0,1]^d$ on which it predicts $+1$. Let $N_w(B)$ be the number of cells in P_w that intersect the boundary of B, denoted by ∂B. For $c_1 > 0$, define the box-counting class $\mathcal{B}_{\text{BOX}}(c_1)$ as the collection of all sets B such that $N_w(B) \leq c_1 w^{d-1}$ for all (sufficiently large) w.

5.2. Problem Set-up

Consider the active learning problem under the following assumptions.

A1 The Bayes optimal classifier B_* resides in $\mathcal{B}_{\text{BOX}}(c_1)$. The corresponding boundary ∂B_* divides $[0,1]^d$ into k connected components[5] (each labeled either $+1$ or -1) and each with volume at least $0 < \beta < 1$. Furthermore, the Hausdorff distance between any two components with the same label is at least $\Delta_1 > 0$.

A2 The marginal distribution of features $P(X = x)$ is uniform over $[0,1]^d$ (the results can be generalized to continuous distributions bounded away from zero).

5. with respect to the usual topology on $\mathbb{R}^d$

A3 The conditional distribution of the label at each feature is bounded away from $1/2$ by a positive margin; i.e., $|P(Y = 1|X = x) - 1/2| \geq \gamma > 0$ for all $x \in [0, 1]^d$.

It can be checked that the set of distributions that satisfy A1-A3 contain the set of distributions BF$(1, 1, C_1, 0, \gamma)$ from Castro and Nowak (2008).

Let G denote the regular square lattice on $[0, 1]^d$ with w vertices equispaced in each coordinate direction. Each vertex is associated with the center of a cell in the partition P_w described above. Figure 3 depicts a case where $d = 2$, $w = 15$ and $k = 2$, where the subset of vertices contained in the set B_* is indicated in red. The minimum distance Δ_1 in A1 above ensures that, for w sufficiently large, the lattice graph also consists of exactly k connected components of vertices having the same label. S^2 can be used to solve the active learning problem by applying it to the lattice graph associated with the partition P_w. In this case, when S^2 requests a (noisy) label of a vertex of G, we randomly draw a feature from the cell corresponding to this vertex and return its label.

Near Minimax Optimality of S^2 .

The main result of this section shows that the excess risk of S^2 for the nonparametric active learning problem described above is minimax-optimal (up to a logarithmic factor). Recall that the excess risk is the difference between the probability of error of the S^2 estimator and the Bayes error rate. The *active learning* minimax lower bound for excess risk is given by $n^{-1/(d-1)}$ (Castro and Nowak, 2008), a significant improvement upon the *passive learning* bound of $n^{-1/d}$ (Scott and Nowak, 2006). Previous algorithms (nearly) achieving this rate required the boundary fragment assumption Castro and Nowak (2008), and so S^2 is near-optimal for a much larger and more realistic range of problems.

Theorem 5 *For any classification problem satisfying conditions A1-A3, there exists a constant $C(k, \beta, \Delta_1, \gamma) > 0$ such that the excess risk achieved by S^2 with n samples on this problem is no more than $C(\frac{\log n}{n})^{\frac{1}{d-1}}$, for n large enough.*

To prove this theorem, we use Theorem 3 to bound the number of samples required by S^2 to be certain (with high probability) of the classification rule everywhere but the cells that are intersected by the boundary ∂B_*. We then use the regularity assumptions we make on the distribution of the features and the complexity of the boundary to arrive at a bound on the excess risk of the S^2 algorithm. This allows us to estimate the excess risk as a function of the number of samples obtained. Refer to Appendix C for the details of the proof.

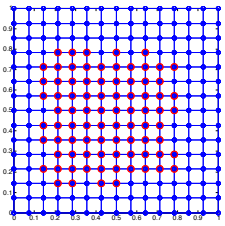


Figure 3: 15×15 grid graph used in experiments. The vertices with the red circle indicate the positive class.

Graph : $(n, \|C\|, \|\partial C\|)$	Mean ∂C-query complexity (10 trials)			
	S^2	AFS	ZLG	BND
Grid : (225, 32 ,68)	**88.8**	160.4	91	192
1 v 2 : (400, 99, 92)	**96.4**	223.2	102.6	370.2
4 v 9 : (400, 457, 290)	**290.6**	367.2	292.3	362.4
CVR : (380, 530, 234)	**235.8**	332.1	236.2	371.1

Table 1: Performance of S^2 , AFS, ZLG, BND.

6. Experiments

We performed some preliminary experiments on the following data sets: (a) **Digits**: This dataset is from the Cedar Buffalo binary digits database originally Hull (1994). We preprocessed the digits by reducing the size of each image down to a 16x16 grid with down-sampling and Gaussian smoothing Le Cun et al. (1990). Each image is thus a 256-dimensional vector with elements being gray-scale pixel values in 0–255. We considered two separate binary classification tasks on this data set: 1 vs 2 and 4 vs 9. Intuitively one might expect the former task to be much simpler than the latter. For each task, we randomly chose 200 digits in the positive class and 200 in the negative. We computed the Euclidean distance between these 400 digits based on their feature vectors. We then constructed a symmetrized 10-nearest-neighbor graph, with an edge between images i, j iff i is among j's 10 nearest neighbors or vice versa. Each task is thus represented by a graph with exactly 400 nodes and about 3000 undirected unweighted edges. Nonetheless, due to the intrinsic confusability, the cut size and the boundary (i.e., edges connecting the two classes) varies drastically across the tasks: 1 vs 2 has a boundary of 92, while 4 vs 9 has a boundary of 290. (b) **Congressional Voting Records (CVR):** This is the congressional voting records data set from the UCI machine learning repository (Bache and Lichman, 2013). We created a graph out of this by thresholding (at 0.5) the Euclidean distance between the data points. This was then processed to retain the largest connected component which had 380 vertices and a boundary size of 234. (c) **Grid:** This is a synthetic example of a 15x15 grid of vertices with a positive core in the center. The core was generated from a square by randomly dithering its boundary. See Figure 3.

We compared the performance of four algorithms: (a) $\mathbf{S^2}$ (b) **AFS** – the active learning algorithm from Afshani et al. (2007); (c) **ZLG** – the algorithm from Zhu et al. (2003b); and (d) **BND** – the experiment design-like algorithm from Gu and Han (2012). We show the number of queries needed before all nodes in ∂C have been queried. This number, which we call ∂C-query complexity, is by definition no smaller than $|\partial C|$. Notice that before completely querying ∂C, it is impossible for any algorithm to guarantee zero error without prior assumptions. Thus we posit that ∂C-query complexity is a sensible measure for the setting considered in this paper. In fact ∂C-query complexity can be thought of as the experimental analogue of the theoretical query complexity of Section 4. These results are shown in Table 1. The bold figures show the best performance in each experiment. As can be seen, S^2 clearly outperforms AFS and BOUND as suggested by our theory. It is quite surprising to see how well ZLG performs given that it was not designed with this objective in mind. We believe that trying to understanding this will be a fruitful avenue for future work.

References

Peyman Afshani, Ehsan Chiniforooshan, Reza Dorrigiv, Arash Farzan, Mehdi Mirzazadeh, Narges Simjour, and Hamid Zarrabi-Zadeh. On the complexity of finding an unknown cut via vertex queries. In *Computing and Combinatorics*, pages 459–469. Springer, 2007.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Sivaraman Balakrishnan, Min Xu, Akshay Krishnamurthy, and Aarti Singh. Noise Thresholds for Spectral Clustering. In *Neural Information Processing Systems*, 2011.

Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

Avrim Blum and Shuchi Chawla. Learning from Labeled and Unlabeled Data using Graph Mincuts. In *International Conference on Machine Learning*, pages 19–26, 2001.

R. Castro and R. Nowak. Minimax bounds for active learning. *IEEE Transactions on Information Theory*, pages 2339–2353, 2008.

Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

Brian Eriksson, Gautam Dasarathy, Aarti Singh, and Robert Nowak. Active Clustering: Robust and Efficient Hierarchical Clustering using Adaptively Selected Similarities. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

Quanquan Gu and Jiawei Han. Towards active learning on graphs: An error bound minimization approach. In *ICDM*, pages 882–887, 2012.

S. Hanneke. Rates of convergence in active learning. *The Annals of Statistics*, pages 333–361, 2011.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5), 1994.

V. Koltchinskii. Rademacher complexities and bounding the excess risk in active learning. *Journal of Machine Learning Research*, pages 2457–2485, 2010.

Y. Le Cun, B. Boser, J. S Denker, D. Henderson, R. E.. Howard, W. Howard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems, 2*, 1990.

S. Minsker. Plug-in approach to active learning. *Journal of Machine Learning Research*, pages 67–90, 2012.

Clayton Scott and Robert D Nowak. Minimax-optimal classification with dyadic decision trees. *Information Theory, IEEE Transactions on*, 52(4):1335–1353, 2006.

L. Wang. Smoothness, disagreement coefficient, and the label complexity of agnostic active learning. *Journal of Machine Learning Research*, pages 2269–2292, 2011.

Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *International Conference on Machine Learning*, pages 912–919, 2003a.

Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. In *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003b.

Acknowledgments

RN is supported in part by the National Science Foundation grant CCF1218189 and the National Institutes of Health grant 1 U54 AI117924-01; XZ is supported in part by National Science Foundation grants IIS 0916038, IIS 0953219, IIS 1216758, and the National Institutes of Health grant 1 U54 AI117924-01.

Appendix A. Proof of Proposition 1

Proposition 1 *Suppose $\mathcal{A}$ is an algorithm that has access to f through a noiseless oracle (i.e., a 0noisy oracle), and suppose that it has a $\epsilon-$query complexity of q, then for each $\gamma \in (0, 0.5)$, there exists an algorithm $\tilde{\mathcal{A}}$ which, using a $\gamma-$noisy oracle achieves a $2\epsilon-$query complexity given by $q \times \left\lceil \frac{1}{2(0.5-\gamma)^2} \log\left(\frac{n}{\epsilon}\right) \right\rceil$.*

Proof Given a $\gamma > 0$, one can design $\tilde{\mathcal{A}}$ as follows. $\tilde{\mathcal{A}}$ simply runs $\mathcal{A}$ as a sub-routine. Suppose $\mathcal{A}$ requires the label of a vertex $v \in V$ to proceed, $\tilde{\mathcal{A}}$ intercepts this label request and repeatedly queries the $\gamma-$noisy oracle r (as defined above) times for the label of v. It then returns the majority vote $\tilde{f}(v)$ as the label of v to $\mathcal{A}$. The probability that such an algorithm fails can be bounded as follows.

$$\mathbb{P}\left[\tilde{\mathcal{A}} \text{ fails after } rq \text{ queries}\right] \leq \mathbb{P}\left[\exists v \in V \text{ s.t. } \tilde{f}(v) \neq f(v)\right] \tag{2}$$

$$+ \mathbb{P}\left[\mathcal{A} \text{ fails after } q \text{ queries} \mid \tilde{f}(v) = f(v), \forall v \in V\right] \tag{3}$$

$$\overset{(a)}{\leq} n \times \mathbb{P}\left[\tilde{f}(v) \neq f(v)\right] + \epsilon \tag{4}$$

$$\overset{(b)}{\leq} n \times e^{-2r(0.5-\gamma)^2} + \epsilon, \tag{5}$$

where (a) follows from the union bound and fact that $\mathcal{A}$ has a $\epsilon-$query complexity of q. (b) follows from applying the Chernoff bound (Chernoff, 1952) to the majority voting procedure : $\mathbb{P}\left[\tilde{f}(v) \neq f(v)\right] = \mathbb{P}\left[\text{Bin}(r, \gamma) \geq 0.5 \times r\right] \leq e^{-2r(0.5-\gamma)^2}$. Therefore, if we set r as in the statement of the proposition, we get the desired result. ■

Appendix B. Proof of Lemma 4

Lemma 4 *Consider a graph $\mathcal{G} = (V, E)$ and a labeling function f with balancedness β. For all $\alpha > 0$, a subset L chosen uniformly at random is a witness to the cut-set with probability at least $1 - \alpha$, as long as $|L| \geq \frac{\log(1/(\beta\alpha))}{\log(1/(1-\beta))}$.*

Proof The smallest component in V is of size at least βn. Let $\mathcal{E}$ denote the event that there exists a component V_i such that $V_i \cap L = \emptyset$ and let $\bar{\beta} = 1 - \beta$. Then, using the union bound and ignoring integer effects, we have $\mathbb{P}[\mathcal{E}] \leq \frac{1}{\bar{\beta}} \cdot \frac{\binom{\bar{\beta}n}{|L|}}{\binom{n}{|L|}} < \frac{\bar{\beta}^{|L|}}{\beta}$, where the last inequality follows from the fact that $\bar{\beta} < 1$.

To conclude, we observe that if we pick $|L|$ as stated in the lemma, then the right-hand side of the above equation drops below α. This concludes the proof. ■

Appendix C. Proof of Theorem 5

Recall that we propose to run S^2 on the lattice graph G corresponding to the partition P_w of $[0,1]^d$. And, recall that when S^2 requests a noisy sample of a vertex in G, a random feature is drawn from the cell that correpsonds to this vertex and its label is returned. Assumption A3 therefore implies that S^2 has access to a $\gamma-$noisy oracle[6]. In what follows, we will derive bounds on the $\epsilon-$query complexity of S^2 in this setting assuming it has access to a noiseless oracle. Then, arguing as in Proposition 1, we can repeat each such query requested by S^2 a total of $\frac{1}{2(0.5-\gamma)^2}\log(w^d/\epsilon)$ times and take a majority vote in order to get the $2\epsilon-$query complexity for a $\gamma-$noisy oracle. So, in the sequel we will assume that we have access to a noise free oracle.

We assume that w is sufficiently large so that $w > \max\{\beta^{-1/d}, 2\Delta_1^{-1}\}$. The first condition is needed since each homogenously labeled component of the problem corresponds to at least βw^d vertices of the lattice graph G and the second condition ensures that there are exactly k connected components in G.

First, we observe that by Lemma 4, if S^2 randomly queries at least $\log(1/\beta\epsilon)/\log(1/(1-\beta))$ vertices, then with probability greater than $1-\epsilon$ it will discover at least one vertex in each of the k connected components. Next, we observe that since there are k connected components of vertices, there are no more than $k^2/4$ cut components in the cut-set[7]. As described in the proof of Theorem 3, once it knows a witness to the cut set, the number of queries made by S^2 can be bounded by adding the number of queries it makes to first discover one cut-edge per each of these cut components to the number of queries needed to perform local search in each of the cut components. Reasoning as in the proof of Theorem 3, for each cut component, S^2 requires no more than $\log w^d$ queries to find one cut edge. To bound the number of queries needed for the local search, we need to bound the size of the boundary $|\partial C|$ and κ, the clusteredness parameter (see Definition 2) of the cut set in G. Towards this end, observe that by assumption A1, there are at most $c_1 w^{d-1}$ cells of the partition P_w that intersects with ∂B_*. Since each cell has at most $d2^{d-1}$ edges, we can conclude that $|\partial C| \leq 2c_1 d(2w)^{d-1}$. To bound κ, let us fix a cut component and note that given one cut edge, there must exist at least one other cut on a two dimensional face of the hypercube containing the first cut edge. Furthermore, this cut edge is contained on a path of length 3 between the vertices of the first cut edge. Since the boundaries of the homogenously labeled components in $[0,1]^d$ are continuous (by definition), it is easy to see that there is an ordering of cut-edges in this cut component $e_1, e_2, \ldots$ such that the distances between them, per Definition 2, satisfy $\delta(e_i, e_{i+1}) = 3$. Since this holds true for all the cut components, this implies that the cut-set is 3-clustered, i.e, $\kappa = 3$. Therefore, the complete boundary can be determined by labeling no more than $(\lceil \log \kappa \rceil + 1)\,|\partial C| = 6c_1 d(2w)^{d-1}$ vertices (since $\lceil \log 3 \rceil = 2$). As observed earlier, if we repeat each of the above queries $\frac{1}{2(0.5-\gamma)^2}\log(w^d/\epsilon)$

6. To be precise, S^2 has access to a $\gamma-$noisy oracle only when querying a vertex whose cell in P_w does not intersect the boundary ∂B_*. However, in the sequel we will assume that S^2 fails to correctly predict the label of any vertex whose cell intersects the boundary, and therefore, this detail does not affect the analysis.
7. Suppose there are z_1 components of label $+1$ and z_2 components of label -1, then there are at most $z_1 z_2$ cut components. The maximum value this can take when $z_1 + z_2 = k$ is $k^2/4$ by the arithmetic mean - geometric mean inequality.

times, we can see that if the number of queries made by S^2 satisfies

$$n \geq \left(6c_1(2w)^{d-1} + \frac{k^2}{4}\log w^d + \frac{\log(1/\beta\epsilon)}{\log(1/(1-\beta))}\right) \times \frac{\log(w^d/\epsilon)}{2(0.5-\gamma)^2}, \tag{6}$$

one can ensure that with probability at least $1-2\epsilon$, S^2 will only possibly make mistakes on the boundary of the Bayes optimal classifier ∂B_*. Let $\mathcal{E}$ be this event and therefore $\mathbb{P}[\mathcal{E}^c] \leq 2\epsilon$.

If we let $S^2(X)$ and $B_*(X)$ denote the prediction of a feature X by S^2 and the Bayes optimal classifier respectively, observe that the excess risk $ER[S^2]$ of S^2 satisfies

$$\begin{aligned} ER[S^2] &= \mathbb{P}\left[S^2(X) \neq Y\right] - \mathbb{P}\left[B_*(X) \neq Y\right] && (7)\\ &\overset{(a)}{\leq} \mathbb{P}[\mathcal{E}^c] + \mathbb{P}\left[S^2(X) \neq Y \middle| \mathcal{E}\right] - \mathbb{P}\left[B_*(X) \neq Y | \mathcal{E}\right] && (8)\\ &\overset{(b)}{\leq} 2\epsilon + \sum_{p \in P_w \cap \partial B_*} \mathbb{P}\left[X \in p\right] && (9)\\ &= \min\{2\epsilon + c_1 2^d w^{-1}, 1\}, && (10) \end{aligned}$$

where (a) follows from conditioning on $\mathcal{E}$. (b) follows from observing first that conditioned on $\mathcal{E}$, $S^2(X)$ agrees with $B_*(X)$ everywhere except on the cells of P_w that intersect the Bayes optimal boundary ∂B_*, and that on this set, we can bound $\mathbb{P}\left[S^2(X) \neq Y | \mathcal{E}\right] - \mathbb{P}\left[B_*(X) \neq Y | \mathcal{E}\right] \leq 1$. The last step follows since by assumption A1, there are at most $2c_1(2w)^{d-1}$ vertices on the boundary, and since, by assumption A2, the probability of a randomly generated feature X belonging to a specific boundary cell is w^{-d}. Therefore, by taking $\epsilon = 1/w$ we have that the excess risk of S^2 is bounded from above by $(2 + c_1 2^d)w^{-1}$ and for this choice of ϵ, the number of samples n satisfies

$$n \geq \left(6c_1(2w)^{d-1} + \frac{k^2}{4}\log w^d + \frac{\log(w/\beta)}{\log(1/(1-\beta))}\right) \times \frac{\log(w^{d+1})}{2(0.5-\gamma)^2}. \tag{11}$$

Finaly, we conclude the proof by observing that if n satisfies (11) and if w is sufficiently large, there exists a constant C which depends only on c_1, k, β, γ, d such that $C\left(\frac{\log n}{n}\right)^{\frac{1}{d-1}} \geq (2 + c_1 2^d)w^{-1}$.

Appendix D. The Tightness of S^2

We will now argue that the upper bounds we derived in the main paper are tight. Towards this end, in what follows, we will assume that a witness (cf. Section 4) to the cut set is known. This allows us to separate the effect of the random sampling phase and the learning phase in such problems.

D.1. Parameter optimality of S^2

In this section, we will show that S^2 is near optimal with respect to the complexity parametrization introduced in Section 4.1. In particular, we will show that given particular values for n, κ, m and $|\partial C|$, there exists a graph such that no algorithm can significantly improve upon S^2 . For what follows, we will set $c \triangleq |\partial C|$. Let us define

$$\mathcal{P} \triangleq \left\{(n, c, m, \kappa) \in \mathbb{N}^4 : m \leq c, c(\kappa+2) \leq n\right\}.$$

We will show that as long the given problem parameters are in $\mathcal{P}$, S^2 is near optimal. While it is trivially true that the number of cut components, m has to be no more than c, the second condition places finer restrictions on the parameter values. These conditions are specific to the construction we present below and can be significantly weakened at the expense of clarity of presentation.

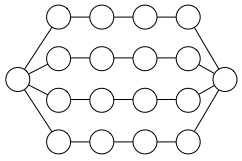


Figure 4: $G(4, 9)$

We will now prove the following theorem.

Theorem 6 *Given a set of values for $n, c, m, \kappa \in \mathcal{P}$, there exists a graph G on n vertices and a set of labelings $\mathcal{F}$ on these vertices such that each $f \in \mathcal{F}$ satisfies:*

- *f induces* no more than *c cuts in the graph.*
- *The number of cut-components is m*
- *Each component is κ-clustered.*

Furthermore, $\log |\mathcal{F}|$ is no smaller than

$$m \log \left(\frac{1}{m} \left\lfloor \frac{n}{\left\lfloor \frac{c}{m} \right\rfloor \left(\left\lfloor \frac{\kappa-1}{2} \right\rfloor \right) + 2} \right\rfloor \times \left(\left\lfloor \frac{\kappa-1}{2} \right\rfloor + 1 \right) \right) + \left(m \left\lfloor \frac{c}{m} \right\rfloor - m \right) \log \left(\left\lfloor \frac{\kappa-1}{2} \right\rfloor + 1 \right).$$

Remark: Notice that $\log |\mathcal{F}|$ is a lower bound on the query complexity of any algorithm that endeavors to learn c cuts in this graph that are decomposed into m components, each of which is $\kappa-$clustered, *even if* the algorithm knows the values m, c, and κ.

Now, this result tells us that if we assume, for the sake of simplicity, that m evenly divides c, κ is odd, and $c(\kappa + 1)$ evenly divides $2nm$ (notice that $c(\kappa + 2) \leq n$ by assumption) then, we have that

$$\begin{aligned} \log |\mathcal{F}| &\geq m \log \left(\frac{1}{m} \left\lfloor \frac{2nm}{c(\kappa+1)} \right\rfloor \times \frac{\kappa+1}{2} \right) + (c-m) \log \left(\frac{\kappa+1}{2} \right) \\ &= m \log \left(\frac{n}{c} \right) + (c-m) \log \left(\frac{\kappa+1}{2} \right). \end{aligned}$$

Comparing this with Theorem 1 in the manuscript, we see that S^2 is indeed parameter optimal.

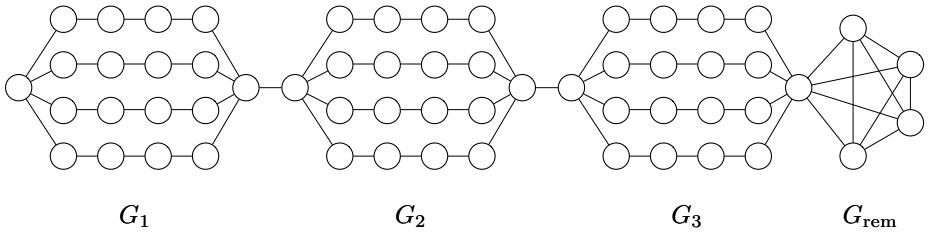


Figure 5: $p = 3$ copies of $G(4, 9)$ linked together

Proof First, define

$$r \triangleq \left\lfloor \frac{c}{m} \right\rfloor,$$
$$k \triangleq 2\left\lfloor \frac{\kappa - 1}{2} \right\rfloor + 1,$$
$$p \triangleq \left\lfloor \frac{2n}{r\,(k-1)+4} \right\rfloor.$$

If $(n, c, m, \kappa) \in \mathcal{P}$, it can be shown that $r \geq 1$ and $p \geq m$. Let $G(r, k)$ denote the following graph on $\frac{r(k-1)+4}{2}$ vertices – two vertices are connected by r edge disjoint paths and each path has $\frac{k-1}{2}$ vertices (and $\frac{k+1}{2}$ edges). This is shown in Fig 4 for $r = 4$ and $k = 9$. G is constructed by linking p copies of $G(r, \kappa)$ and a "remainder" graph G_{rem} which is a clique with $n - p\left(r\left(\frac{k-1}{2}\right) + 2\right)$ as shown in Fig 5. We will denote these p copies of $G(r, k)$ as $G_1, \ldots, G_p$.

Let $\mathcal{F}$ be the set of all labelings obtained as follows:

1. Choose m out of the p subgraphs $G_1, \ldots, G_p$ without replacement. There are $\binom{p}{m}$ ways to do this.

2. In each of these m chosen subgraphs, pick r edges to be cut edges, one on each of the r paths. There are $\left(\frac{k+1}{2}\right)^r$ ways to do this in one subgraph. Therefore, there is a total number of $\left(\frac{k+1}{2}\right)^{mr}$ ways to do this.

3. Now, let the left most vertex of G_1 be labeled $+1$, the rest of the labels are completely determined by the cut-set we just chose.

Notice that for each $f \in \mathcal{F}$, the following holds: (a) there are exactly m cut-components in the graph, (b) the number of cuts is $m \times \left\lfloor \frac{c}{m} \right\rfloor \leq c$, and (c) in each cut component, the cuts are $k \leq \kappa$ close.

The total number of such labelings is given by:

$$\binom{p}{m} \times \left(\frac{k+1}{2}\right)^{mr} \geq \left(\frac{p}{m}\right)^m \left(\frac{k+1}{2}\right)^{rm}.$$

Therefore, we can lower bound $\log|\mathcal{F}|$ as follows

$$\begin{aligned}
\log|\mathcal{F}| &\geq m\log\left(\frac{p}{m}\right) + mr\log\left(\frac{k+1}{2}\right) \\
&= m\log\left(\frac{1}{m}\left\lfloor\frac{2n}{r(k-1)+4}\right\rfloor\right) + m\left\lfloor\frac{c}{m}\right\rfloor\log\left(\left\lfloor\frac{\kappa-1}{2}\right\rfloor+1\right) \\
&= m\log\left(\frac{1}{m}\left\lfloor\frac{2n}{\left\lfloor\frac{c}{m}\right\rfloor 2(\left\lfloor\frac{\kappa-1}{2}\right\rfloor)+4}\right\rfloor\right) + m\left\lfloor\frac{c}{m}\right\rfloor\log\left(\left\lfloor\frac{\kappa-1}{2}\right\rfloor+1\right) \\
&= m\log\left(\frac{1}{m}\left\lfloor\frac{n}{\left\lfloor\frac{c}{m}\right\rfloor(\left\lfloor\frac{\kappa-1}{2}\right\rfloor)+2}\right\rfloor\left(\left\lfloor\frac{\kappa-1}{2}\right\rfloor+1\right)\right) \\
&\qquad + \left(m\left\lfloor\frac{c}{m}\right\rfloor - m\right)\log\left(\left\lfloor\frac{\kappa-1}{2}\right\rfloor+1\right).
\end{aligned}$$

This concludes the proof. ■

D.2. Two Dimensional Grid

In this section, we will show that in the case of the 2-dimensional grid S^2 is near optimal even if we fix the graph before hand. Notice that in this sense, this result is stronger than the one presented in the previous section and is particularly relevant to the theory of nonparametric active learning. Consider the example of a 2-dimensional $r \times r$ grid, where the bottom-left vertex is labeled $+1$ and the top-right vertex is labeled -1 (see Fig. 5). We want to count/lower bound the total number of labelings such that there are exactly 2 connected components and such that the cut-size of the labeling is no more than r. Notice that the logarithm of this number will be a lower bound on the query complexity of any algorithm that endeavors to learn a cut set of size no more than r.

Theorem 7 *The number of ways to partition an $r \times r$ grid into 2 components using a cut of size at most r such that the bottom-left vertex and top-right vertex are separated is lower bounded by 2^r.*

Proof We will first restrict our attention to cuts of size exactly r. Consider the grid shown in Figure 6. We will begin by making a few simple observations. Each of the $(r-1)\times(r-1)$ boxes that contains a cut, has to contain at least 2 cuts. Furthermore, since these cuts are only allowed to partition the grid into 2 connected components, cuts are located in contiguous boxes. Therefore, there are at most $r-1$ boxes that contain cuts.

We will think of a cut as a *walk* on the grid of $(r-1)\times(m-1)$ boxes (labeled as $(i,j), 1 \leq i,j \leq r-1$ in Fig 6) and lower bound the total number of such walks. Observe that for a walk to correspond to a valid cut, it must contain one of the boxes labeled S and one of the boxes labeled T. By symmetry, it suffices only consider walks that originate in an S box and end in a T box.

To lower bound the number of valid walks, we are going to restrict ourselves to *positive walks* – walks that only move either right (R) or down (D). Notice that such walks traverse exactly $r-1$ boxes. Towards this end, we first observe that there are $2(r-1)$ such walks each of which originates in an S-block and terminates at the diametrically opposite T-block. These walks are made up of entirely R moves or entirely of D moves. Therefore, to count the number of remaining walks, we can restrict our attention to walks that contain at least one R move and one D move. Notice that

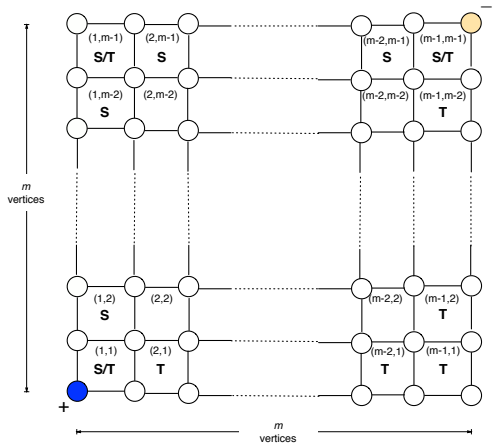


Figure 6: $r \times r$ grid with bottom-left and top-right vertex oppositely labeled.

such walks cannot cross over the diagonal. Therefore, by symmetry, it suffices to consider walks that start in an S-box on the left column: $\{(1,1),\ldots,(1,r-1)\}$ and end in a T-box in the bottom row: $\{(1,1),(2,1),\ldots,(r-1,1)\}$. Suppose, for $j \geq 2$, we start a walk at block $(1,j)$, then the walk has to make exactly $j-1$ down moves and $(m-2-j+1)$ right moves (since the total number of blocks in the walk is $r-1$). Therefore, the total number of such positive walks that originate from $(1,j)$ is $\binom{m-2}{j-1}$. Reasoning similarly, we conclude that there are $\sum_{j=2}^{r-3}\binom{r-2}{j-1}$ such positive walks from one of the S-boxes in the left column to one of the T-boxes in the bottom row. Finally, observe that the walk that starts at $(1,1)$ and ends at $(1,m-1)$ correspond to two different cuts since the $(1,r-1)$ box has two valid edges that can be cut. Similarly the walk $(1,r-1)-\cdots-(r-1,r-1)$ corresponds to 2 valid cuts. Therefore, the total number of cuts from such walks is given by $2(r-1)+2(2+\sum_{j=2}^{r-3}\binom{r-2}{j-1}) = 2(r-1)+2^{r-1}$, where the multiplication by 2 inside follows from the symmetry we used.

Observe now that if we allow the cuts to be smaller than r, then the total number of cuts is given by summing over the cut-sizes as follows: $\sum_{i=2}^{r} 2(i-1)+2^{i-1} = \binom{r-1}{2} + 2^r - 2 \geq 2^r$. This concludes the proof. ■

Therefore, any algorithm will need to submit at least $\log(2^r) = \mathcal{O}(r)$ queries before it can discover a cut of size at most r and in fact, from the proof above, this seems like a pretty weak lower bound (since we restricted ourselves only to positive walks). However, observe that since $\kappa = 3$ and $|\partial C| \leq r$ here, Theorem 1 (from the manuscript) tells us that S^2 submits no more than $\mathcal{O}(r)$ queries for the same. Extending this argument to other families of graphs is an interesting avenue for future work.