Efficient Version Space Algorithms for "Human-in-the-Loop" Model Development

Luciano Di Palma, Yanlei Diao, Anna Liu

Journal of Machine Learning Research 21 (2020) 1-49 Submitted 9/18; Published 9/20

Efficient Version Space Algorithms for "Human-in-the-Loop" Model Development

Luciano Di Palma LUCIANO.DI-PALMA@POLYTECHNIQUE.EDU
Yanlei Diao YANLEI.DIAO@POLYTECHNIQUE.EDU
Laboratoire d'Informatique de l'Ecole Polytechnique
Ecole Polytechnique
Palaiseau, 91120, France

Anna Liu ANNA@MATH.UMASS.EDU
Dept. of Mathematics and Statistics
University of Massachusetts Amherst
Amherst, MA 01003, United States

Editor:

Abstract

When active learning (AL) is applied to help the user develop a model on a large dataset through interactively presenting data instances for labeling, existing AL techniques can suffer from two main drawbacks: first, they may require hundreds of labeled data instances in order to reach high accuracy; second, retrieving the next instance to label can be time consuming, making it incompatible with the interactive nature of the human exploration process. To address these issues, we introduce a novel version space based AL algorithm for kernel classifiers, which not only has strong theoretical guarantees on performance, but also allows for an efficient implementation in time and space. In addition, by leveraging additional insights obtained in the user labeling process, we are able to factorize the version space to perform active learning in a set of subspaces, which further reduces the user labeling effort. Evaluation results show that our algorithms significantly outperform state-of-the-art version space algorithms, as well as a recent factorization-aware algorithm, for model development over large data sets.

Keywords: active learning, version space, data exploration, factorization, rounding

1. Introduction

Active learning (Tong and Koller, 2001; Trapeznikov et al., 2011; Settles, 2012; Gonen et al., 2013) has been studied intensively to address the problem of learning classification models with limited training data. An active learner can examine the current classification model and develop a series of inquiries from the data source to obtain new labeled instances that increase classification accuracy as fast as possible, hence reducing the overall burden of sample acquisition and labeling.

This line of work has recently gained increased interest in industry, in a trend known as "Machine Learning for Everyone": IT companies are delivering cloud platforms such as Amazon SageMaker[1] and Google AutoML[2] to help every data user to develop machine

1. `https://aws.amazon.com/sagemaker/`
2. `https://cloud.google.com/automl/`

learning models from their data sets with minimum effort. A key question, however, is how to obtain a high-quality training data set with minimum user effort. While industry solutions are limited to manual labeling by a dedicated IT team or crowdsourcing, recent research on interactive data exploration (IDE) for model development (Dimitriadou et al., 2014, 2016; Liu et al., 2017; Huang et al., 2018) brings active learning to bear on the new process of model learning with minimum user effort and under interactive performance: In this setting, the user aims to build a classification model over a data set from no or very few labeled instances. Active learning is applied to select a minimum sequence of data instances for the user to label in order to derive an accurate model, while at the same time, offering interactive performance in presenting the next data instance for the user to review and label. As such, IDE presents a more challenging setting for active learning than traditional model learning: besides minimizing labeling effort, it further poses a performance requirement when scanning the underlying data set to identify best data instances for labeling next.

Existing active learning techniques, however, often fail to provide satisfactory performance when such models need to be built over large data sets. For example, our evaluation results show that on a Sloan Digital Sky Survey (SDSS) data set of 1.9 million data instances, existing active learning techniques (Tong and Koller, 2001; Settles, 2012; Gonen et al., 2013), as well as a recent active learning based IDE system (Huang et al., 2018) requires the user to label over 200 instances in order to learn a classification model over 6 attributes with F-score of 80%.[3] Asking the user to label a large number of instances to achieve high accuracy, known as the slow convergence problem, is undesirable.

In this work, we refer to the slow increase of F-score in the early labeling iterations in active learning as the *cold start* problem. We aim to devise new techniques to overcome the cold start problem in order to expedite convergence while providing interactive performance for user labeling. Our design of new techniques embodies two ideas:

Version Space Algorithms. First, we would like to leverage Version Space (VS) algorithms (Tong and Koller, 2001; Dasgupta, 2005; Golovin and Krause, 2011; Settles, 2012) because they present a strong theoretical foundation for convergence. These algorithms model all possible configurations of a classifier that can correctly classify the current set of labeled data as a set of hypotheses forming a *version space* $\mathcal{V}$, and aim to seek the next instance such that its acquired label will enable $\mathcal{V}$ to be reduced. The best known example is the bisection rule (Dasgupta, 2005; Golovin and Krause, 2011), which among all unlabeled instances, looks for the one whose label allows $\mathcal{V}$ to be reduced by half or most close to that. It is shown theoretically to have near-optimal performance for convergence.

Implementing the bisection rule, however, is prohibitively expensive due to the exponential size of a version space. To reduce cost, various approximations have been proposed. Some of the most popular techniques are Simple Margin (Tong and Koller, 2001), Query-by-Disagreement (Settles, 2012), Kernel Query-by-Committee (KQBC) (Gilad-Bachrach et al., 2006), and ALuMA (Gonen et al., 2013). The first three methods often suffer from sub-optimal performance since they are very rough approximations of the bisection rule. On the other hand, ALuMA can better approximate the bisection rule by sampling the version space, which, however, can be very costly to run. For example, ALuMA incurs a high

3. F-score is a better measure for exploring a large data set than classification error given that the user interest, i.e., the positive class, often covers only a small fraction of the data set; classifying all instances to the negative class has low classification error but poor F-score.

time cost in each iteration by running a hit-and-run sampling procedure with thousands of steps to sample a single hypothesis and repeating the procedure to sample hundreds of hypotheses, as well as a high space cost by computing a kernel matrix over the entire data set. Therefore, our work develops new techniques to enable VS algorithms to achieve both fast convergence and high efficiency on large data sets.

Factorization. To make VS algorithms practical for large data sets, our second idea is to augment them with additional insights obtained in the user labeling process. In particular, we observe that often when a user labels a data instance, the decision making process can be broken into a set of smaller questions, and the answers to these questions can be combined to derive the final answer. For example, when a customer decides whether a car model is of interest, she may have a set of questions in mind: "Is gas mileage good enough? Is the vehicle spacious enough? Is the color a preferred one?". While the user may have high-level intuition that each question is related to a subset of attributes (e.g., the space depends on the body type, length, width, and height), she is not able to specify these questions precisely. It is because she may not know the exact threshold value or the exact relation between a question and related attributes (e.g., how the space can be specified as a function of body type, length, width, and height). The above insight allows us to design new version space algorithms that leverage the high-level intuition a user has for breaking the decision making process, formally called a *factorization structure*, to combat the cold start problem.

More specifically, we make the following contributions in this paper:

1. *A new theoretical framework for version space algorithms over kernel classifiers* (Section 3): We propose a new theoretical framework that allows for an efficient implementation of the Generalized Binary Search (Golovin and Krause, 2011) strategy over kernel classifiers. Compared to previous works (Gilad-Bachrach et al., 2006; Gonen et al., 2013), our work offers both strong theoretical guarantees on performance and an efficient implementation in time and space. Our key techniques include a dimensionality reduction theorem that restricts the kernel matrix computations to the set of labeled points, and a Cholesky decomposition-based technique to efficiently estimate the version space reduction of any sample. We also prove generalization error bounds on accuracy and F-score, enabling our techniques to run over a sample from the original large data set with minimal performance loss.

2. *Implementation and Optimizations* (Section 4): Based on our theoretical results, we propose an optimized VS algorithm called OptVS. Similarly to the works of Gilad-Bachrach et al. (2006) and Gonen et al. (2013), OptVS uses the hit-and-run algorithm (Lovász, 1999) for sampling the version space. However, hit-and-run may require thousands of iterations to output a high-quality sample, which can incur a high time cost. To reduce the cost, we develop a range of sampling optimizations to improve both sample quality and running time. In particular, we provide a highly efficient version of the *rounding* technique (Lovász, 1986) for improving the sample quality from the version space.

3. *A Factorized Version Space Algorithm* (Section 5): Additionally, we propose a new algorithm that leverages the factorization structure provided by the user to create subspaces, and factorizes the version space accordingly to perform active learning in

the subspaces. Compared to recent work (Huang et al., 2018) that also used factorization for active learning, our work explores it in the new setting of VS algorithms and eliminates the strong assumptions made such as convexity of user interest patterns, resulting in significant performance improvement while increasing the applicability in real-world problems. We also provide theoretical results on the optimality of our factorized VS algorithm.

4. *Evaluation* (Section 6): Using two real-world data sets and a large suite of user interest patterns, our evaluation results show that (1) for lower dimensional problems, our optimized VS algorithm, without factorization, already outperforms existing VS algorithms including Simple Margin (Tong and Koller, 2001), Query-by-Disagreement (Settles, 2012), and ALuMA (Gonen et al., 2013); (2) for higher dimensional problems, our factorized VS algorithm outperforms VS algorithms (Tong and Koller, 2001; Settles, 2012; Gonen et al., 2013), as well as DSM (Huang et al., 2018), a factorization-aware algorithm, often by a wide margin while maintaining interactive speed. For example, for a complex user interest pattern tested, our algorithm achieves F-score of over 80% after 100 iterations of labeling, while DSM is still at 40% and all other VS algorithms are at 10% or lower.

2. Related Work

In this section, we survey most relevant work in active learning and data exploration for model development.

Active Learning. The recent results on active learning are surveyed in Settles (2012). Our work focuses on a common form called pool-based active learning. In this scenario, there is a small set of labeled data $\mathcal{L}$ and a large pool of unlabeled data $\mathcal{U}$ available. In active learning, an example is chosen from the pool in a greedy fashion, according to a utility measure used to evaluate all instances in the pool (or, if $\mathcal{U}$ is large, a subsample thereof). In our problem setting, the labeled data $\mathcal{L}$ is what the user has provided thus far. The pool $\mathcal{U}$ is a subsample of size m of the unlabeled part of the data set. The utility measure varies with the classifier and algorithm used.

In practice, AL is usually employed in domains where labeling data is expensive and labeled data sets are scarce. For example, recent domains of application include biomedical image segmentation (Yang et al., 2017), health monitoring (Bull et al., 2018), and crisis management (Pohl et al., 2018). Active learning has also been applied in crowd-sourcing scenarios (Song et al., 2018; Hantke et al., 2017), where multiple annotators work on the same labeling task. In our work, we intent to apply active learning, in particular the version space algorithms, in the setting of interactive data exploration for model development.

Version Space Algorithms. Version space (VS) algorithms are a particular class of active learning procedures. In such a procedure, the learner starts with a set of possible classifiers (hypotheses), which we denote by $\mathcal{H}$. The version space $\mathcal{V}$ is defined as the subset of classifiers $h \in \mathcal{H}$ consistent with the labeled data, i.e. $h(x) = y$ for all labeled points (x, y). As new labeled data is obtained, the set $\mathcal{V}$ will shrink, until we are left with a single hypothesis. Various strategies have been developed to select new instances for labeling.

- *Generalized Binary Search* (Dasgupta, 2005; Golovin and Krause, 2011): Also called the *Version Space Bisection* rule, this algorithm searches for a point x for which the classifiers in $\mathcal{V}$ disagree the most; that is, the sets $\mathcal{V}^{x,y} = \{h \in \mathcal{V} : h(x) = y\}$ have roughly the same size, for all possible labels y. It has strong theoretical guarantees on convergence: the expected number of iterations needed to reach 100% accuracy is at most a constant factor larger than the optimal algorithm on average. Implementing the bisection rule, however, is prohibitively expensive: for each unlabeled instance x in the data set, one must evaluate $h(x)$ for each hypothesis h in the version space, which is exponential in size $O(N^d)$, where N is number of unlabeled instances and d is the VC dimension (Mohri et al., 2012). A number of approximations of the bisection rule have been introduced in the literature:

- *Simple Margin* (Tong and Koller, 2001): As a rough approximation of the bisection rule for SVM classifiers, it leverages an heuristic that data points close to the SVM's decision boundary closely bisect the version space, $\mathcal{V}$. However, it can suffer from suboptimal performance, specially when $\mathcal{V}$ is very asymmetric.

- *Kernel Query-by-Committee* (KQBC) (Gilad-Bachrach et al., 2006) and *Query-by-Disagreement* (QBD) (Settles, 2012): These algorithms approximate the bisection rule by constructing two representative hypotheses in the version space, and then select any data point for which these two classifiers disagree. In particular, KQBC selects two random hypotheses from the version space, while QBD trains a positively biased and a negatively biased classifiers. Again, these methods can suffer from suboptimal performance since the selected point is only guaranteed to "cut" $\mathcal{V}$, but possibly not by a large amount.

- *ALuMA* (Gonen et al., 2013): ALuMA is an application of the Generalized Binary Search (Golovin and Krause, 2011) algorithm for linear classifiers, and uses a sampling technique for estimating the version space reduction of any sample. It can also support kernels via a preprocessing step, but this procedure can be very expensive to run. It is shown to outperform several other version space algorithms, including Simple Margin and Query-by-Committee (Seung et al., 1992). Hence, we use ALuMA as a baseline version space algorithm in this work.

More recently, version space based techniques have also been successfully applied to a broader range of scenarios, such as cost-sensitive classification (Krishnamurthy et al., 2017) and batch-mode active learning (Chen and Krause, 2013). We focus on how version space techniques can be applied to the problem of interactive data exploration for model development, which we further describe below.

Data Programming under Weak Supervision. One form of data exploration for model development is the *data programming* framework introduced by Snorkel (Ratner et al., 2016; Bach et al., 2017), which has gained attraction in recent years. In this framework, an expert user writes a set of *labeling functions* representing simple heuristics used for labeling data instances. By treating these labeling functions as weak learners, Snorkel is capable of building an accurate classifier without having the user manually label any data instance. In more recent work, a data programming system called Snuba (Varma and

Ré, 2018) is designed to automatically generate labeling functions from an initial labeled data set, leading to an improvement in classification accuracy and further reduction of user manual effort. However, the initial labeled data set is still required to be on the order of hundreds of labeled instances, which is nontrivial to obtain, especially in the new setting of "Machine Learning for Everyone" where an everyday user may start with no labeled data at all.

Interactive Data Exploration for Model Development. In the domain of interactive data exploration for model development, a main objective is to design a system that guides the user towards discovering relevant data points in a large data set. One line of work is "explore-by-example" systems (Dimitriadou et al., 2014, 2016; Huang et al., 2018; Liu et al., 2017), which leverage active learning to obtain a small set of user labeled data points for building an accurate model of the user interest. These systems require minimizing both the user labeling effort and running time in each iteration to find a new data point for labeling. In particular, recent work (Huang et al., 2018) is shown to outperform prior work (Dimitriadou et al., 2014, 2016) via the following technique:

Dual-Space Model (DSM) (Huang et al., 2018): In this work, the user interest pattern P is assumed to form a convex object in the data space. For such a pattern, this work proposes a "dual-space model" (DSM), which builds not only a classifier but also a polytope model of the data space $\mathcal{D}$, offering information including the areas known to be positive and areas known to be negative. It uses both the polytope model and the classifier to decide the best instance to choose for labeling next. In addition, DSM explores *factorization* in a limited form: when the user pattern P is a conjunction of sub-patterns on non-overlapping attributes, it factorizes the data space into low-dimensional spaces and runs the dual-space model in each subspace.

3. Generalized Binary Search over Kernel Classifiers

In this section, we address the problem of how to efficiently realize the Generalized Binary Search (GBS) algorithm for kernel classifiers. Although a few works (Gonen et al., 2013; Gilad-Bachrach et al., 2006) have also attempted to apply version space based techniques to kernel classifiers, their approaches fail to scale to large data sets.

In Gonen et al. (2013), they have developed a novel algorithm, ALuMA, for applying the GBS strategy over linear classifiers. This algorithm comes with several theoretical results on convergence speed and label complexity guarantees. ALuMA also supports an extension to kernel classifiers by first running a preprocessing step over the data. However, this procedure has two major drawbacks: first, it requires computing the kernel matrix (and its Cholesky decomposition) over all data points, which is time and memory inefficient for large data sets; second, this preprocessing step requires an upper bound on the total Hinge loss of the best separator, which usually is not know in practice.

In Gilad-Bachrach et al. (2006), they proposed an extension of the Query-by-Committee (Seung et al., 1992) algorithm to kernel classifiers, called Kernel Query-by-Committee (KQBC). They also demonstrated a dimensionality reduction theorem which avoids computing the kernel matrix over all points, improving the algorithm's efficiency. However, estimating whether each data point "cuts" the version space or not requires running a separate sampling procedure, which can become particularly expensive to do in the pool-based settings. Another drawback of this work is the lack of theoretical guarantees on convergence speed.

In what follows, we introduce a new version space based technique over kernel classifiers. Compared to previous work, our approach shares the same near-optimal guarantees on convergence as GBS, while still allowing for an efficient implementation in both time and space similar to KQBC, thus being compatible with the interactive data exploration scenario.

3.1 Generalized Binary Search Algorithm Overview

Let $\mathcal{X} = \{x_i\}_{i=1}^N$ be the collection of unlabeled data points, and let $y_i \in \mathcal{Y}$ represent the unknown label of x_i. The set $\mathcal{Y}$ is called the *label space*, which is assumed to be finite. The user interest pattern can be modeled by an *hypothesis function* $h : \mathcal{X} \to \mathcal{Y}$, with h belonging to some *hypothesis space* $\mathcal{H}$. This space is assumed to to satisfy the following condition:

Axiom 1 (Realizability) *There exists a classifier $h^* \in \mathcal{H}$ matching the user preference, that is, achieving zero classification error.*

With this, our objective is to identify the hypothesis h^* while minimizing the number of instances labeled by the user.

In the above formulation, it is possible to exist more than one hypothesis $h^* \in \mathcal{H}$ matching the user interest. One well-known solution (Dasgupta, 2005; Golovin and Krause, 2011; Gonen et al., 2013) for this ambiguity problem is to define an equivalence relation over $\mathcal{H}$ which identifies any two hypotheses having the same predictions over $\mathcal{X}$:

$$h \sim h' \iff h(x_i) = h'(x_i),\, \forall x_i \in \mathcal{X}$$

With this, our original problem becomes finding the equivalence class $[h^*]$, which is unique. Additionally, we assume a known probability distribution $\pi([h])$ over the set of equivalence classes $\hat{\mathcal{H}} := \mathcal{H} / \sim$, representing our prior knowledge over which hypotheses are more likely to match the user's preferences. In the absence of prior knowledge, an uniform prior can be assumed.

The *version space* is the set of all hypotheses consistent with a labeled set $\mathcal{L}$: $\mathcal{V} = \{[h] \in \hat{\mathcal{H}} : h(x) = y,\, \forall (x, y) \in \mathcal{L}\}$. Additionally, let's introduce the shorthand $\mathcal{V}^{x,y} = \mathcal{V} \cap \{[h] \in \hat{\mathcal{H}} : h(x) = y\}$. Then, the version space bisection rules searches for the point x such that all the sets $\mathcal{V}^{x,y}$ have approximately the same probability mass:

Definition 2 (Generalized Binary Search - GBS) *Let $\mathcal{V}$ denote the version space at any iteration of the AL process. For any $(x, y) \in \mathcal{X} \times \mathcal{Y}$, we define the* ***cut probabilities****:*

$$p_{x,y} = \mathbb{P}(h(x) = y \,|\, [h] \in \mathcal{V}) = \frac{\pi(\mathcal{V}^{x,y})}{\pi(\mathcal{V})} \tag{1}$$

Then, the GBS strategy selects any unlabeled point $x^* \in \mathcal{U}$ satisfying

$$x^* \in \operatorname*{arg\,max}_{x \in \mathcal{U}} 1 - \sum_{y \in \mathcal{Y}} p_{x,y}^2$$

It has been shown (Golovin and Krause, 2011) that the above strategy enjoys a near-optimal performance guarantee. Let $cost(\mathcal{A})$ denote the average number of queries an active learner $\mathcal{A}$ takes to identify a random hypothesis drawn from π. Then, the GBS algorithm satisfies

$$cost(GBS) \leq OPT \cdot \left(1 + \ln \frac{1}{\min_h \pi([h])}\right)^2 \tag{2}$$

where $OPT = \min_{\mathcal{A}} cost(\mathcal{A})$.

3.2 Parameterized Hypothesis Space

In the next sections, we would like to work with classifiers parameterized by some vector θ (for example linear classifiers). In these cases, it is preferable to work directly with the parameters themselves instead of the classifiers, specially when devising algorithms.

First of all, we define what is a "parameter":

Definition 3 (Parameterization) *We call a function $\psi : \Omega \ni \theta \to h_\theta \in \mathcal{H}$ a parameterization of $\mathcal{H}$ over the parameter space $\Omega \subset \mathbb{R}^D$ if the map $\Omega \ni \theta \to [h_\theta] \in \hat{\mathcal{H}}$ is surjective.*

We also define a *parameterized version space* over Ω, which is the set of parameters θ whose corresponding classifiers h_θ are consistent with the labeled data:

Definition 4 (Parameterized Version Space) *Let $\psi : \Omega \to \mathcal{H}$ be a parameterization, and let $\mathcal{L}$ be a labeled set of points. We define the parameterized version space as*

$$\mathcal{V}_\theta = \{\theta \in \Omega : h_\theta(x) = y, \forall (x, y) \in \mathcal{L}\}$$

One advantage of working over the parameter space is we have more freedom in choosing a prior distribution π over the hypothesis classes. To see this, let $p_0(\theta)$ be any probability distribution over Ω, representing a prior over parameters. Then, we can define π as the push-forward distribution:

$$\pi([h']) \coloneqq \mathbb{P}_\theta(h_\theta \sim h') \tag{3}$$

With this, we can now define an equivalent GBS strategy directly over the parameter space Ω:

Theorem 5 *Let $\psi : \Omega \to \mathcal{H}$ be a parametrization, and let $p_0(\theta)$ be any prior over Ω. Let's denote by GBS_θ the greedy strategy selecting*

$$\arg\max_{x \in \mathcal{U}} 1 - \sum_{y \in \mathcal{Y}} p_{x,y}^2$$

where $p_{x,y} = \mathbb{P}(h_\theta(x) = y \mid \theta \in \mathcal{V}_\theta)$. Then, this strategy satisfies

$$cost(GBS_\theta) \leq OPT \cdot \left(1 + \ln \frac{1}{\min_h \pi([h])}\right)^2$$

where $OPT = \min_{\mathcal{A}} cost(\mathcal{A})$ and π is as in Equation (3).

3.3 Kernel Classifiers

With the above, we are in measure of applying the GBS strategy over kernel classifiers. First, let's restrict ourselves to the *binary case*, and set $\mathcal{Y} = \{\pm 1\}$. Additionally, let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a *positive-definite kernel function*,[4] with corresponding *feature map* $\phi : \mathbb{R}^d \to \mathcal{F}$. Our objective is to apply the GBS algorithm to the set of linear classifiers over $\mathcal{F}$:

$$\mathcal{H} = \{h_{b,f} : h_{b,f}(x) = \text{sign } b + \langle \phi(x), f \rangle_{\mathcal{F}}, \text{ for } (b, f) \in \Omega\}$$

where $\Omega = \{(b, f) \in \mathbb{R} \times \mathcal{F} : b^2 + \|f\|_{\mathcal{F}}^2 \leq 1\}$ is the parameter space. The constraint $b^2 + \|f\|_{\mathcal{F}}^2 \leq 1$ can be included without any loss in generalization power, since $h_{b,f} \sim h_{rb,rf}$ for any $r > 0$.

One major problem in dealing with these classifiers is dimensionality: for some choices of kernel, $\mathcal{F}$ is infinite-dimensional (Mohri et al., 2012), and thus inconvenient to work with. However, the result below allows us to restrict our parameters to a finite-dimensional subspace of $\mathcal{F}$:

Lemma 6 *Let $S = span(\phi(x_1), \ldots, \phi(x_N))$ and let's denote by $f|_S$ the orthogonal projection of $f \in \mathcal{F}$ onto S. Then, for all $f \in \mathcal{F}$ and all $x_i \in \mathcal{X}$ we have:*

$$\langle f, \phi(x_i) \rangle_{\mathcal{F}} = \langle f|_S, \phi(x_i) \rangle_{\mathcal{F}}$$

In particular, this implies that $h_{b,f} \sim h_{b,f|_S}$, for all $(b, f) \in \mathbb{R} \times \mathcal{F}$.

With this, we can restrict ourselves to the set of parameters on the form $f = \sum_{i=1}^N \alpha_i \phi(x_i)$, which leads to the following definition:

Definition 7 (Kernel Classifiers) *Let K be the $N \times N$ kernel matrix $K_{ij} = k(x_i, x_j)$. The set of kernel classifiers is defined to be*

$$\mathcal{H}_\alpha = \{h_{b,\alpha} : h_{b,\alpha}(x) = \text{sign } b + \sum_{i=1}^N \alpha_i k(x_i, x), \text{ for } (b, \alpha) \in \Omega_\alpha\}$$

where $\Omega_\alpha = \{(b, \alpha) \in \mathbb{R} \times \mathbb{R}^N : b^2 + \alpha^T K \alpha \leq 1\}$ is the corresponding parameter space.

Now, we wish to apply the GBS strategy over the parameter space Ω_α. Let $p_0(b, \alpha)$ be any prior over the parameters. Given some labeled data $\mathcal{L}$, the parameterized version space is by definition

$$\mathcal{V}_\alpha = \{(b, \alpha) \in \Omega_\alpha : y_i(b + \alpha^T K_i) \geq 0, \forall (x_i, y_i) \in \mathcal{L}\}$$

where K_i is the i-th row of K. With this, the cut probabilities are given by

$$p_{x_j,\pm} = \mathbb{P}(h_{b,\alpha}(x_j) = \pm 1) = \mathbb{P}\left(\pm(b + K_j^T \alpha) > 0\right)$$

where (b, α) is a random variable drawn from the conditional distribution $p_0|_{\mathcal{V}_\alpha}$. In what follows, we focus on how to efficiently compute the probabilities $p_{x_j,\pm}$.

4. Any kernel k can be made positive-definite by adding a small perturbation: $k'(x, y) = k(x, y) + \lambda \mathbb{1}(x = y)$, with $\lambda > 0$

3.4 Dimensionality Reduction

One major problem in computing the cut probabilities $p_{x_j,\pm}$ is the high-dimensionality of Ω_α: since the number of data points N is usually very large, each $p_{x_j,\pm}$ will be very expensive to estimate. Below, we show a method for reducing the dimensionality from $N+1$ to $|\mathcal{L}|+2$, which is a more tractable range of values.

We start with a simplifying assumption:

Assumption for this section: In what follows, we assume the labeled set $\mathcal{L}$ is composed by the first t data points $\{x_1, \ldots, x_t\}$; this can always be assumed w.l.g., since it amounts to a simple re-indexing of data points. In addition, we assume a uniform prior $p_0(b, \alpha)$ over the parameter space Ω_α of kernel classifiers.

Our first step is to consider the Cholesky decomposition $K = LL^T$, with L being lower-triangular. By defining the change of variables $\beta = L^T\alpha$, the probability computations can be slightly simplified to

$$p_{x_j,\pm} = \mathbb{P}\left(\pm(b + \beta^T L_j) > 0\right)$$

where L_j is the j-th row of L. In addition, since L^T is invertible and we are assuming a uniform prior over Ω_α, (b, β) must also follow the uniform distribution over the set

$$\mathcal{V}_\beta = \{(b, \beta) \in \Omega_\beta : y_i(b + \beta^T L_i) \geq 0,\ 1 \leq i \leq t\} \tag{4}$$

where Ω_β is the unit ball in $\mathbb{R}^{N+1}$. With this, we can apply the following dimensionality reduction lemma, which is an extension of Theorem 1 from Gilad-Bachrach et al. (2006):

Lemma 8 *Define $S_j = \mathbb{R} \times span(L_1, \ldots, L_t, L_j)$, and let (b', β') be drawn uniformly over $\mathcal{V}_\beta \cap S_j$. Then, the cut probabilities $p_{x_j,\pm}$ satisfy*

$$p_{x_j,\pm} = \mathbb{P}\left(\pm(b' + \beta'^T L_j) > 0\right) \tag{5}$$

With this result, the dimensionality is effectively reduced from $N + 1$ to $t + 2$, and algorithms should be much more efficient to run. However, there is still a slight problem: the vectors L_j live in a N-dimensional space, and they can be very expensive to compute. In order to fix this, we first construct an orthonormal basis for S_j:

Lemma 9 *Let's denote by L_j^k the entry of L at row j and column k. Then, for any $j > t$ the set*

$$\left\{e_1, e_2, \ldots, e_{t+1}, \frac{T_j}{\|T_j\|}\right\}$$

is an orthonormal basis for S_j, where e_i is the i-th canonical vector in $\mathbb{R}^{N+1}$ and $T_j = (\vec{0}_{t+1}, L_j^{t+1}, \ldots, L_j^N)$.

Now we have an orthonormal basis, we can represent any vector $(b', \beta') \in S_j$ as a sum

$$(b', \beta') = \sum_{i=1}^{t+1} w_i e_i + w_{t+2}\frac{T_j}{\|T_j\|} \tag{6}$$

where $w \in \mathbb{R}^{t+2}$ are the orthonormal basis coefficients. By replacing this representation on Equations (4) and (5), we obtain our main result:

Definition 10 (Partial Cholesky Factor) *Let $\mathcal{L} = \{(x_1, y_1), \dots, (x_t, y_t)\}$ be the labeled set, and let $K = LL^T$ be the Cholesky decomposition of the $N \times N$ kernel matrix over all data points. We define the partial Cholesky factor $\ell_i \in \mathbb{R}^{t+2}$ as*

$$\ell_i = \left(1, L_i^1, \dots, L_i^t, \sqrt{K_{ii} - \sum_{k=1}^{t} (L_i^k)^2}\right) \tag{7}$$

In particular, we note that $\ell_i^r = 0$ whenever $i \leq t$ and $r \geq i + 2$.

Theorem 11 *Let w be a random vector drawn uniformly over the set*

$$W_t = \{w \in \mathbb{R}^{t+2} : \|w\| \leq 1 \text{ and } y_i w^T \ell_i \geq 0,\, 1 \leq i \leq t\} \tag{8}$$

where ℓ_i represents the i-th partial Cholesky factor. Then, the cut probabilities $p_{x_j,\pm}$ satisfy

$$p_{x_j,\pm} = \mathbb{P}\left(\pm\, w^T \ell_j > 0\right)$$

With this, we have shown that the cut probabilities $p_{x_j,\pm}$ depend on two quantities: the partial Cholesky factor ℓ_j, and a random sample $w \sim Unif(W_t)$. In particular, our formulation has three major advantages compared to previous works. First, we do not need to compute the whole Cholesky matrix L; instead, only the first t columns are enough (as opposed to Gonen et al. 2013). Second, the sample $w \sim Unif(W_t)$ does not depend on the particular training point x_j we are estimating (unlike Gilad-Bachrach et al. 2006), which translates into a more efficient estimation of the cut probabilities. Third, the Cholesky property allows us to introduce a novel sampling optimization called *rounding cache*, which significantly speeds-up our sampling procedure. A more complete discussion on these points will be left to Section 4.

3.5 Approximations for Large Data Sets

In some cases, the data set $\mathcal{X}$ can be so large it does not fit in memory, or working with $\mathcal{X}$ directly is too costly. In these cases, we propose the *subsampling* procedure below:

1. First, a random sample $S \subset \mathcal{X}$ is obtained
2. Then, the AL procedure is run over S , returning a classifier $h \in \mathcal{H}$ achieving low-error over S
3. Finally, h will be used for labeling the entire $\mathcal{X}$

The success of the above procedure relies on appropriately choosing the size of the sample S. In one hand, choosing a small sample can greatly reduce the computational cost of the AL procedure. On the other hand, if S is too small then the classifier h will probably not generalize well to the entire $\mathcal{X}$, specially for more complex hypothesis classes. This delicate balance in choosing the appropriate sample size is captured by the following lemma, which is a simple application of Theorem 2.2 from Mohri et al. (2012):

Definition 12 *Let $\mathcal{X}$ be a data set, and let h^* be the target classifier. Then, for any classifier h and any $S \subset \mathcal{X}$, we define the* ***accuracy error*** *of h over S to be*

$$\epsilon_{acc}(h, S) = \frac{1}{|S|} \sum_{x \in S} \mathbb{1}(h(x) \neq h^*(x))$$

Lemma 13 *Let $\mathcal{X} = \{x_1, \ldots, x_N\}$ be a large data set and let $S = \{X_i\}_{i=1}^m$ be a sample with replacement from $\mathcal{X}$. Additionally, consider any hypothesis space $\mathcal{H}$ satisfying the realizability axiom, and let h^* be the target classifier. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ we have*

$$\forall h \in \mathcal{H},\ \epsilon_{acc}(h, \mathcal{X}) \leq \epsilon_{acc}(h, S) + \sqrt{\frac{\log |\hat{\mathcal{H}}| + \log \frac{2}{\delta}}{m}}$$

where $\hat{\mathcal{H}} = \mathcal{H} / \sim$ is the set of equivalence classes defined in Section 3.1.

From the above lemma, irrespectively of which classifier $h \in \mathcal{H}$ the AL procedure computes, it is guaranteed to achieve low error over $\mathcal{X}$ under two conditions:

- *Enough data points are labeled by the user*: as more data points are labeled, the AL algorithm is able to compute an increasingly more accurate classifier h over S. In particular, we note that choosing a fast-converging AL algorithm let us achieve low error rates with minimal labeling effort.
- *The sample size is large enough*: m should be chosen to control the complexity of our hypothesis class. In particular, for a desired error rate η we need to sample at least

$$m = \frac{\log |\hat{\mathcal{H}}| + \log \frac{2}{\delta}}{\eta^2}$$

  data points. We also note that although $\hat{\mathcal{H}}$ is usually exponentially large, it can still lead to an acceptable sample size is many cases. For example, if $\mathcal{H}$ has finite VC-dimension D then $|\hat{\mathcal{H}}| = \mathcal{O}(|\mathcal{X}|^D)$ (Mohri et al., 2012), resulting in a logarithmic dependency of m on the data set size.

Now, let's place ourselves in the IDE scenario. Due to the extreme class imbalance of these cases, it would be ideal to also obtain generalization bounds for other performance metrics than accuracy, such as precision, recall, and F-score. In fact, we can extend the above results to these metrics as well:

Definition 14 (Precision, Recall, and F-score errors) *For any $r, s \in \{-, +\}$, let's define the confusion matrix entry by*

$$C^{r,s}(h, S) = \frac{1}{|S|} \sum_{x \in S} \mathbb{1}(h(x) = r, h^*(x) = s)$$

With this, we define the following error quantities

$$\epsilon_{prec}(h, S) = 1 - precision(h, S) = \frac{C^{+,-}}{C^{+,+} + C^{+,-}}$$
$$\epsilon_{rec}(h, S) = 1 - recall(h, S) = \frac{C^{-,+}}{C^{+,+} + C^{-,+}}$$
$$\epsilon_{fscore}(h, S) = 1 - fscore(h, S) = \frac{C^{+,-} + C^{-,+}}{2C^{+,+} + C^{+,-} + C^{-,+}}$$

Theorem 15 *Let h be any classifier. Then, we have*

$$\epsilon_{fscore}(h, S) \leq \max(\epsilon_{prec}(h, S), \epsilon_{rec}(h, S)) \leq \frac{1}{\mu}\epsilon_{acc}(h, S)$$

where $\mu = \mathbb{P}(Y = 1)$ is the positive class selectivity.

By combining the above result with Theorems 13, it is possible achieve similar error guarantees in precision, recall, and F-score by simply having a sample $\frac{1}{\mu^2}$-times larger.

3.5.1 The Majority Vote Classifier

In the particular case of version space algorithms, it is usually not specified how to choose a low-error classifier h. This is because the procedure is assumed to be run until a single hypothesis remains in the version space, and consequently all the labels are known. However, the lack of a suitable convergence criteria and the high cost of manual labeling make this procedure impractical. In the literature, different methods were devised to overcome this limitation, such as: training a SVM classifier over the labeled set (Tong and Koller, 2001), choosing a random h in the version space (Gilad-Bachrach et al., 2006), or, more recently, computing a *majority vote* of classifiers sampled from the version space (Gonen et al., 2013). In our work, we also adopt the majority voting idea:

Definition 16 (Majority Vote classifier) *Let $\mathcal{X} = \{x_1, \ldots, x_N\}$ be a data set, and let $\mathcal{H}$ be a set of classifiers. Also, assume a probability distribution π over the set of equivalence classes $\mathcal{H} / \sim$. The majority vote classifier is defined as*

$$MV^\pi(x) = \arg\max_{y \in \mathcal{Y}} p^\pi_{x,y}$$

where $p^\pi_{x,y} = \mathbb{P}_{[H]\sim\pi}(H(x) = y)$.

The MV classifier has a very intuitive definition: for any data point, its predicted label is an agreement between the predictions of all classifiers in $\mathcal{H}$, weighted by π. In the particular case of VS algorithms, π can be chosen as the restriction of the prior probability to the version space, which makes $p^\pi_{x,y}$ coincide with the cut probability definition (1).

One advantage of the MV classifier is it has interesting generalization properties:

Theorem 17 *For any $S \subset \mathcal{X}$ and any distribution π, the MV^π classifier satisfies*

$$\epsilon_{acc}(MV^\pi, S) \leq 2\mathbb{E}_{[H]\sim\pi}[\epsilon_{acc}(H, S)]$$

In particular, under the conditions of Theorem 13, with probability $1-\delta$ *it holds*

$$\epsilon_{acc}(MV^{\pi}, \mathcal{X}) \leq 2\mathbb{E}_{[H]\sim\pi}[\epsilon_{acc}(H, S)] + \sqrt{\frac{2}{m}\left(\log|\hat{\mathcal{H}}| + \log\frac{2}{\delta}\right)}$$

From the above result, we can see that the MV classifier possesses similar generalization error bounds as any $h \in \mathcal{H}$, but it only depend on the *average training error* according to π. In the particular case of VS algorithms, as more points are labeled, the version space shrinks and π becomes more and more concentrated around the target classifier $[h^*]$, which consequently reduces the average training error.

4. Implementation and Optimizations

In this section, we detail how to efficiently implement the results of Theorem 11, resulting in an efficient versions space based active learner called OptVS. At the core of our selection strategy and label prediction (majority voting) lies the computation of the cut probabilities $p_{x,\pm}$, which can be done via Algorithm 1 below. In what follows, we give a precise description of each step in this algorithm.

Algorithm 1 Computing the cut probabilities

```
Input: labeled set $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^t$, unlabeled set $\mathcal{U} \subset \{x \in \mathcal{X} : (x, \pm 1) \notin \mathcal{L}\}$, sample size
    $M$, any ellipsoid $\mathcal{E}_0 \supset W_t$, kernel function $k$
Output: The cut probabilities $p_{x_*,+}$, for $x_* \in \mathcal{U}$
 1: $K_{ij} \leftarrow k(x_i, x_j)$, $1 \leq i, j \leq t$
 2: $L \leftarrow Cholesky(K)$
 3: $\ell_i \leftarrow \left(1, L_i^1, \dots, L_i^t, 0\right)$, $1 \leq i \leq t$
 4: $W_t \leftarrow \{w \in \mathbb{R}^{t+2} : \|w\| \leq 1$ and $y_i \ell_i^T w \geq 0$, $1 \leq i \leq t\}$
 5: $\mathcal{E}_{round} \leftarrow rounding\_algorithm(W_t, \mathcal{E}_0)$
 6: $\{w^i\}_{i=1,\dots,M} \leftarrow hit\_and\_run\,(W_t, M, \mathcal{E}_{round})$
 7: for all $x_* \in \mathcal{U}$ do
 8:     $K_* \leftarrow [k(x_1, x_*), \dots, k(x_t, x_*)]^T$
 9:     $L_* \leftarrow L^{-1} K_*$
10:     $\ell_* \leftarrow \left(1, L_*^1, \dots, L_*^t, \sqrt{k(x_*, x_*) - \|L_*\|^2}\right)$
11:     $p_{x_*,+} \leftarrow \frac{1}{M}\sum_{i=1}^M \mathbb{1}\left(\ell_*^T w^i > 0\right)$
12: end for
13: return $\{p_{x_*,+}$, for $x_* \in \mathcal{U}\}$
```

4.1 Computing the Partial Cholesky Factors ℓ_j

From Theorem 11, the first step in estimating $p_{x_j,+}$ is to compute the *partial Cholesky factor* ℓ_j, as defined in Equation (7). In particular, this vector can be easily computed once we have $\tilde{L}_j = (L_j^1, \dots, L_j^t)$, where L is the Cholesky decomposition of the kernel matrix K. Now, from the equation $K = LL^T$ it easily follows that

$$LL_j = K_j \Rightarrow \tilde{L}\tilde{L}_j = \tilde{K}_j$$

where $\tilde{L}$ is the upper-left $t \times t$ submatrix of L and $\tilde{K}_j$ is the t first components of K_j. With this, we note two things:

1. $\tilde{L}$ corresponds to the Cholesky factor of $\tilde{K}$, the kernel matrix over the labeled data points (lines 1 and 2 in the pseudo-code).

2. Since $\tilde{L}$ is lower-triangular, the system $\tilde{L}\tilde{L}_j = \tilde{K}_j$ can be efficiently solved via forward substitution (lines 8 and 9 in the pseudo-code).

Finally, assuming a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of complexity $\mathcal{O}(d)$ (which is the case for most popular kernel functions), computing the partial Cholesky factors incurs a complexity of $\mathcal{O}\left(t^2(d+t) + |\mathcal{U}|t(d+t)\right)$.

4.2 Estimating the Cut Probabilities via Sampling

Since the probabilities $p_{x_j,+}$ have no closed-formula expression, we estimate these quantities via a *sampling procedure* (Gonen et al., 2013; Gilad-Bachrach et al., 2006). This step corresponds to lines 6 and 11 in Algorithm 1.

First, by the Law of Large Numbers we can write

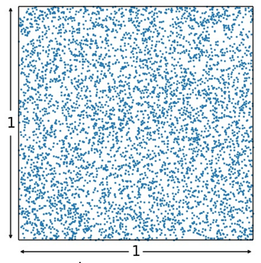
$$p_{x_j,+} \approx \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left(\ell_j^T w^i > 0\right)$$

where $\{w^i\}_{i=1,\dots,M}$ is an i.i.d. sample following the uniform distribution over W_t. In fact, sampling uniformly over convex bodies like W_t is a well-known problem in convex geometry, for which we can apply the hit-and-run algorithm (Lovász, 1999). Hit-and-run generates a Markov Chain inside W_t which converges to the uniform distribution; in other words, the longer the hit-and-run chain we generate, the closer to uniformly distributed our sample will be.

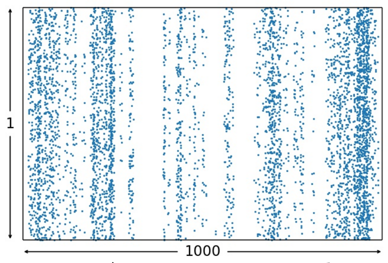
However, there is one issue with this sampling procedure: since we are assuming $\{w^i\}_{i=1}^M$ to be an independent sample, a separate hit-and-run procedure is needed for generating each w^i, which can become quite expensive to run. In order to overcome this overhead, we opt for a slightly different sampling procedure that requires a single chain to be generated. Since the hit-and-run chain $\{w^0, w^1, \dots\}$ forms a positive Harris-recurrent Markov Chain (Bélisle et al., 1993), the Law of Large Numbers theorem for Markov Chains (Kaspi et al., 1997, Theorem 17.1.7) guarantees that

$$p_{x_j,+} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathbb{1}\left(\ell_j^T w^n > 0\right)$$

Thus, estimating $p_{x_j,+}$ can be done through a single chain. Further, we employ two well-known techniques for improving the mixing time (Harms and Roebroeck, 2018; Link and Eaton, 2012): *Burn-in* consists of discounting the first B samples from the chain, since their distribution is very far from the stationary one; *Thinning*, on the other hand, only keeps one sample every T iterations after burn-in, in hopes to reduce sample correlation. In the end, we return $\{w^B, w^{B+T}, \dots, w^{B+(M-1)T}\}$ as the final sample.



(a) A 1×1 square



(b) A 1000×1 rectangle

Figure 1: Distribution of the first 5000 samples generated by the hit-and-run procedure over a square and a long rectangle. As we can see, the samples distribution is closer to uniform for evenly elongated (or "round") convex bodies, thus implying a smaller mixing time.

The technical details on how hit-and-run is implemented will be left to Appendix B. In particular, we note that the hit-and-run sampling has a complexity of $\mathcal{O}(Bt^2 + MTt^2)$, while the cut probability estimation takes $\mathcal{O}(|\mathcal{U}|Mt)$.

4.3 Improving the Sample Quality via Rounding

The hit-and-run algorithm can have a large mixing time, specially in cases where the convex body $W \subset \mathbb{R}^n$ is very elongated in one direction. In practice, this can lead to a poor sample quality, whose distribution is far away from uniform even after thousands of steps; see Figure 1 for an example of this behavior.

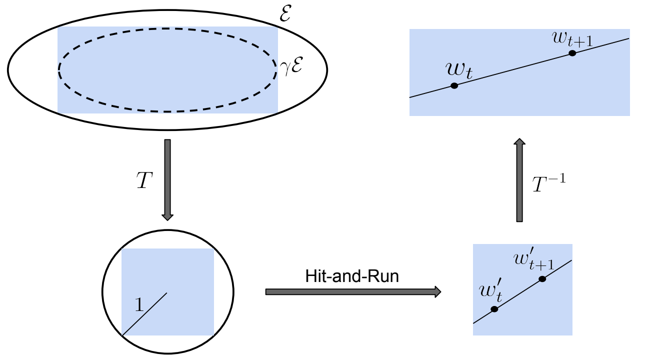
In order to address the problem above, we adopt a *rounding procedure* (Lovász, 1986; De Martino et al., 2015). In general terms, rounding is a preprocessing algorithm which attempts to make W more evenly elongated across all directions via a linear transformation T. Once such a transformation is found, the hit-and-run algorithm is run over $T(W)$, and a sample over W is finally obtained via the inverse transformation T^{-1}; see Figure 2(a) for an illustration of this process.

One well-known method for computing a rounding transformation is via Lovász algorithm (Lovász, 1986). Let $\mathcal{E}(z, P) = \{w \in \mathbb{R}^n : (w - z)^T P^{-1}(w - z) \leq 1\}$ denote an ellipsoid with *center* z and *scaling matrix* P.[5] The main idea is to find a *rounding ellipsoid* $\mathcal{E}$ satisfying

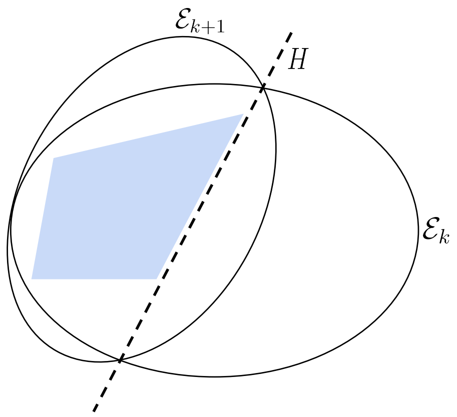
$$\gamma\mathcal{E} \subset W \subset \mathcal{E} \tag{9}$$

where $\gamma\mathcal{E} = \mathcal{E}(z, \gamma^2 P)$ is obtained by scaling the axes of $\mathcal{E}$ by a factor $\gamma > 0$. If γ is large enough, then $\mathcal{E}$ should approximately capture the directions of major elongation of W; thus, by choosing T as a linear transformation mapping $\mathcal{E}$ into an unit ball, any major stretching of W should also be corrected, and the hit-and-run chain should mix quickly; refer to Appendix C for more details. In particular, we note that the resulting hit-and-run chain is guaranteed to mix in $\mathcal{O}^*(n^2/\gamma^2)$ steps.

5. The scaling matrix must be symmetric and positive-definite.



(a) A rounding transformation T corrects the elongation of the convex body (in blue) before generating the hit-and-run samples.



(b) Illustration of the rounding ellipsoid algorithm. A bounding ellipsoid $\mathcal{E}_k$ is repeatedly shrunk via hyperplane cuts H.

Figure 2: Illustration of rounding procedure over a convex body (in blue).

All that remains now is how to compute a rounding ellipsoid. The main idea behind Lovász's algorithm is to construct an approximation of the *minimum volume ellipsoid* $\mathcal{E}^*$ containing W. The reason is that $\mathcal{E}^*$ is known to satisfy $\frac{1}{n}\mathcal{E}^* \subset W \subset \mathcal{E}^*$; thus, by computing an ellipsoid close enough to $\mathcal{E}^*$, one could hope to obtain a γ factor close to $1/n$ as well.

An outline of the ellipsoid computation algorithm is presented in Algorithm 2. Starting from any ellipsoid $\mathcal{E}_0 \supset W$, the algorithm constructs a sequence $\{\mathcal{E}_1, \mathcal{E}_2, \ldots\}$ which gets step-by-step closer to $\mathcal{E}^*$. More precisely, given $\mathcal{E}_k \supset W$, the algorithm proceeds as follows:

1. First, the algorithm checks whether $\gamma\mathcal{E}_k \subset W$ holds true for a given threshold $0 < \gamma < 1/n$. If so, $\mathcal{E}_k$ is returned as rounding ellipsoid.

2. Otherwise, it constructs a *separating half-space* $H(a,b) = \{x : a^T x \leq b\}$ satisfying $W \subset H(a,b)$ and $\alpha(\mathcal{E}_k, H) \leq -\gamma$, where

$$\alpha(\mathcal{E}, H) = \frac{a^T z - b}{\sqrt{a^T P a}}$$

3. Finally, we set $\mathcal{E}_{k+1}$ as *minimum volume ellipsoid* containing $\mathcal{E}_k \cap H(a,b)$, which can be analytically computed; see Figure 2(b) for an illustration, and check Algorithm 6 for more details.

The above properties ensure that this algorithm will always terminate in a finite number of steps. This is because of two factors: first, $W \subset H$ implies that $W \subset \mathcal{E}_k$ for all k; second, the property $\alpha(\mathcal{E}_k, H) \leq -\gamma$ guarantees (Bland et al., 1981) that $vol(\mathcal{E}_{k+1}) < c \cdot vol(\mathcal{E}_k)$, where $0 < c < 1$ is a constant depending only on γ and n. More precisely, these factors ensure an upper bound of

$$\frac{\log\left(vol(\mathcal{E}_0)/vol(W)\right)}{\log\left(1/c\right)} \tag{10}$$

Algorithm 2 Ellipsoid computation algorithm

```
Input: convex body $W \subset \mathbb{R}^n$, ellipsoid $\mathcal{E}_0 \supset W$, threshold $0 < \gamma < 1/n$
Output: An ellipsoid $\mathcal{E}$ satisfying $\gamma\mathcal{E} \subset W \subset \mathcal{E}$
1: $k \leftarrow 0$
2: while $\gamma\mathcal{E}_k \not\subset W$ do
3:     $H \leftarrow get\_separating\_halfspace(\mathcal{E}_k, W)$
4:     $\mathcal{E}_{k+1} \leftarrow minimum\_volume\_ellipsoid(\mathcal{E}_k, H)$
5:     $k \leftarrow k + 1$
6: end while
7: return $\mathcal{E}_k$
```

in the number of iterations until the rounding ellipsoid is found.

Despite the generality and guarantees of the rounding algorithm, it can be very costly to run in practice, specially as the dimensionality increases. This is specially problematic for our version space algorithm: every time we receive a new labeled point, not only the dimensionality of W_t increases, but W_t itself is modified and a new rounding ellipsoid must be computed. Thus, in order to improve its efficiency, we propose two major optimizations:

1. *Ellipsoid caching* (Section 4.3.1): In practice, we note that the rounding algorithm can take a high number of iterations to converge. To address this issue, we introduce a novel *ellipsoid caching* procedure that re-uses previous rounding ellipsoids in order to compute a $\mathcal{E}_0$ of small volume.

2. *Optimized ellipsoid algorithm* (Section 4.3.2): The algorithm proposed by Lovász (1986) was conceived for general convex bodies. However, its generality comes with a price: round-off errors introduced by every rounding iteration tend to accumulate, sometimes leading to numerical instability issues. We introduce a novel rounding algorithm that leverages the particular format of W_t and allows for a numerically stable implementation.

4.3.1 Ellipsoid caching algorithm

In order to tackle the high number of rounding iterations, we introduce an *ellipsoid caching* procedure. Based on Equation (10), one way of reducing the number of iterations is to find an initial ellipsoid $\mathcal{E}_0$ of volume as small as possible. With this idea in mind, our strategy is to construct $\mathcal{E}_0 \supset W_{t+1}$ by re-using the rounding ellipsoid $\mathcal{E}_{rnd} \supset W_t$ from a previous labeling iteration. Since $\mathcal{E}_{rnd}$ is an approximation of the minimum volume ellipsoid, $\mathcal{E}_{rnd}$ and W_t should be close in volume, which gives us hope in constructing $\mathcal{E}_0$ of volume close to W_{t+1} as well.

First, let's recall that $W_t = \{w \in \mathbb{R}^{t+2} : \|w\| \leq 1 \text{ and } y_i w^T \ell_i \geq 0,\ 1 \leq i \leq t\}$ as defined in Equation (8), where t is the number of labeled points. When a new labeled point is obtained, the set W_t is subject to three modifications:

1. The dimension increases by one

2. Each ℓ_i is appended with a 0 to the right, for $1 \leq i \leq t$

3. A new linear constraint $y_{t+1}\ell_{t+1}^T w \geq 0$ is included

In order to simplify our computation, we aim to find an ellipsoid $\mathcal{E}_0 \supset W_{t+1}$ that is independent of the new constraint ℓ_{t+1} added. In particular, the above properties guarantee that it is enough to find $\mathcal{E}_0$ containing $W_t \times [-1, 1] \supset W_{t+1}$. With this insight, we can easily prove the following result:

Lemma 18 *Let $\mathcal{E}(z, P)$ be any ellipsoid containing W_t. Then, the ellipsoid $\mathcal{E}_0(z', P')$ given by*

$$z' = \begin{bmatrix} z \\ 0 \end{bmatrix} \qquad P' = (t+3)\begin{bmatrix} \frac{1}{t+2}P & 0 \\ 0 & 1 \end{bmatrix}$$

contains $W_t \times [-1, 1]$, and thus can be used as the starting ellipsoid for the rounding algorithm.

4.3.2 An optimized rounding algorithm

In the original ellipsoid computation algorithm of Lovász (1986), checking if $\gamma\mathcal{E} \subset W$ requires computing the extreme points of $\mathcal{E}$, a process that uses the eigenvalues and eigenvectors of the scaling matrix P. However, due to round-off errors introduced by each rounding iteration, P may no longer be positive-definite, making it infeasible to compute the extreme points. In order to avoid these numerical issues, we introduce a new ellipsoid computation algorithm for convex bodies on the form $W = \{w \in \mathbb{R}^n : \|w\| \leq 1 \text{ and } a_i^T w \leq 0,\ 1 \leq i \leq m\}$ which avoids the diagonalization step, and allows for a numerically stable implementation.

In our version of the ellipsoid algorithm, we follow the same procedure outlined in Algorithm 2. In particular, this means that there are two main steps to consider: how to check if $\gamma\mathcal{E} \subset W$ and, if that is not the case, find a half-space H satisfying $\alpha(\mathcal{E}, H) > -\gamma$. In order to tackle the first problem, we rely on the following result:

Lemma 19 *Consider the convex body $W = \{w \in \mathbb{R}^n : \|w\| \leq 1$ and $a_i^T w \leq 0,\ 1 \leq i \leq m\}$, and let $\mathcal{E}(z, P)$ be an ellipsoid. Then, for any $\gamma > 0$ we have*

$$\gamma\mathcal{E} \subset W \iff \max_{1 \leq i \leq m} \alpha_i \leq -\gamma \text{ and } \max_{\|w\|=1} \alpha_w \leq -\gamma$$

where $\alpha_i = \alpha(\mathcal{E}, H(a_i, 0)) = z^T a_i / \sqrt{a_i^T P a_i}$ and $\alpha_w = \alpha(\mathcal{E}, H(w, 1)) = (z^T w - 1)/\sqrt{w^T P w}$.

The above result also helps with the solution for the second part of the algorithm, that is, finding a cutting half-space $H(a, b)$ satisfying $\alpha(\mathcal{E}, H) > -\gamma$. First, let's define $i^* = \arg\max_i \alpha_i$ and $w^* = \arg\max_{\|w\|=1} \alpha_w$. If $\gamma\mathcal{E} \not\subset W$ then either $\alpha_{i^*} > -\gamma$ or $\alpha_{w^*} > -\gamma$, meaning that either $H(a_{i^*}, 0)$ or $H(w^*, 1)$ can be chosen as cutting half-space.

Now, the only point remaining is how to compute $\max_{\|w\|=1} \alpha_w$. This is a non-linear, constrained optimization problem, for which many efficient solvers exist (SLSQP, COBYLA, etc). However, since calling the solver can be expensive if done repeatedly, we introduce two small optimizations:

- Only call the solver if $\alpha_{i^*} \leq -\gamma$: in other words, we use the cuts $H(a_i, 0)$, which are relatively cheap to compute, until no longer possible.

Algorithm 3 Optimized rounding algorithm

```
Input: convex body $W = \{w \in \mathbb{R}^n : \|w\| \leq 1 \wedge a_i^T w \leq 0, 1 \leq i \leq m\}$, ellipsoid $\mathcal{E}_0 = \mathcal{E}(z_0, P_0) \supset W$, threshold $0 < \gamma < \frac{1}{n}$
Output: An ellipsoid $\mathcal{E}$ satisfying $\gamma\mathcal{E} \subset W \subset \mathcal{E}$
1:  $k \leftarrow 0$
2:  while True do
3:      $\alpha_{i^*} \leftarrow \max_i \frac{z_k^T a_i}{\sqrt{a_i^T P_k a_i}}$
4:      $\alpha_z \leftarrow \frac{\|z_k\|(\|z_k\|-1)}{\sqrt{z_k^T P_k z_k}}$
5:      if $\alpha_{i^*} \geq \alpha_z$ and $\alpha_{i^*} > -\gamma$ then
6:          $\mathcal{E}_{k+1} \leftarrow$ minimum_volume_ellipsoid$(\mathcal{E}, H(a_{i^*}, 0))$
7:      else if $\alpha_z \geq \alpha_{i^*}$ and $\alpha_z > -\gamma$ then
8:          $\mathcal{E}_{k+1} \leftarrow$ minimum_volume_ellipsoid$(\mathcal{E}, H(z_k, \|z_k\|))$
9:      else
10:         $\alpha_{w^*} \leftarrow \max_{\|w\|=1} \frac{w^T z_k - 1}{\sqrt{w^T P_k w}}$
11:         if $\alpha_{w^*} \leq -\gamma$ then
12:             return $\mathcal{E}_k$
13:         end if
14:         $\mathcal{E}_{k+1} \leftarrow$ minimum_volume_ellipsoid$(\mathcal{E}_k, H(w^*, 1))$
15:     end if
16:     $k \leftarrow k + 1$
17: end while
```

- Only call the solver if $\alpha_z = \alpha(\mathcal{E}, H(z, \|z\|)) \leq -\gamma$: the cut $H(z, \|z\|)$ has the interesting property that $\alpha_z < 0 \iff \|z\| < 1$. In particular, the condition $\alpha_z \leq -\gamma < 0$ allows us to avoid calling the solver in the cases where $z \notin W$.

Finally, by putting together all of the above considerations, we are now ready to present our own rounding algorithm for W, as described in Algorithm 3.

One last point to discuss is the numerical stability of our rounding algorithm. The major point of concern is computing the quantity $1/\sqrt{a^T P a}$ for some vector $a \neq 0$. As we previously discussed, round-off errors may cause P_k to no longer be positive-definite, which can lead to $a^T P a \leq 0$. In fact, the same kind of problem is known to occur within the minimum volume ellipsoid routine, for which a few solutions have already been proposed in the literature (Bland et al., 1981; Goldfarb and Todd, 1982).

Our method of choice is to make use of the LDL^T decomposition of P (Goldfarb and Todd, 1982). By doing so, we can write $a^T P a = \sum_i D_{ii}(L_i^T a)^2$, which is guaranteed to be positive. Additionally, the minimum volume ellipsoid routine can be modified to directly update the matrices L and D in a numerically stable way, and we no longer have to store the scaling matrix P; more details on how to implement this step are deferred to Appendix C. We also note that although this procedure could also be applied to Lovász's rounding algorithm, we would still have to store or compute the scaling matrix P since it is necessary for the diagonalization step. In contrast, our solution removed the diagonalization step and completely eliminates the dependency on P via the decomposition, thus solving the numerical instability issue.

4.4 Hit-and-run's Starting Point

Hit-and-run starts by finding a point w^0 inside W_t. Although this could be done by solving a linear programming task, it can take a significant amount of time. Instead, we rely on the rounding procedure: since the computed ellipsoid $\mathcal{E}(z, P)$ satisfies $\gamma\mathcal{E} \subset W_t$ for some $\gamma > 0$, in particular it guarantees that $z \in W_t$, which can be used as a starting point for hit-and-run.

5. A Factorized Version Space Algorithm

To improve the efficiency of version space (VS) algorithms on large data sets, we aim to augment them with additional insights obtained in the user labeling process. In particular, we observe that often when a user labels a data instance, the decision making process can be broken into a set of smaller "yes" or "no" questions, which can be answered independently, and the answers to these questions can be combined to derive the final answer. Let's consider the following example: when a customer decides whether a car model is of interest, she can have the following questions in mind:

Q_1: Is gas mileage good enough?
Q_2: Is the vehicle spacious enough?
Q_3: Is the color a preferred one?

We do not expect the user to specify her questions precisely as classification models (which requires knowing the exact shape of the decision function and all constants used), but rather to have a high-level intuition of the set of questions and to which attributes each question is related.

Factorization Structure. Formally, we model such an intuition of the set of questions and the relevant attributes as a factorization structure: Let us model the decision making process using a complex question Q defined on an attribute set $\boldsymbol{A}$ of size d. Based on the user intuition, Q can be broken into smaller independent questions, $Q_1, \ldots, Q_K$, where each Q_k is posed on a subset of attributes $\boldsymbol{A}^k = \{A_{k1}, \cdots, A_{kd_k}\}$. The family of attribute sets, $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^K)$, $|\boldsymbol{A}^1 \cup \cdots \cup \boldsymbol{A}^K| \leq d$ may be disjoint, or overlapping in attributes as long as the user believes that decisions for these smaller questions can be made independently. $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^K)$ is referred to as the *factorization structure*.

Note that the factorization structure needs to be provided by the user as it reflects her understanding of her own decision making process. The independence assumption in the decision process should not be confused with the data correlation issue. For example, the color and the size of cars can be statistically correlated, e.g., large cars are often black in color. But the user decision does not have to follow the data characteristics; e.g., the user may be interested in large cars that are red. As long as the user believes that her decision for the color and that for the size are independent, the factorization structure ({color}, {size}) applies. To the contrary, if the two attributes are not independent in the decision making process, e.g., the user prefers red if the car is small and black if the car is large, but the year of production is an independent concern, then the factorization structure can be ({color size}, {year}).

Decision functions. Given a data instance x, we denote $x^k = proj(x, \boldsymbol{A}^k)$ the projection of x over attributes $\boldsymbol{A}^k$. We use $Q_k(x^k) \to \{-,+\}$ to denote user's decision on x^k. Then we assume that the final decision from the answers to these questions is a boolean function $F : \{-,+\}^K \to \{-,+\}$:

$$Q(x) = F(Q_1(x^1), \ldots, Q_K(x^K)) \tag{11}$$

In this work, we assume that the decision function F is given by the user. The most common example is the conjunctive form, $Q_1(x^1) \wedge \ldots \wedge Q_K(x^K)$, meaning that the user requires each small question to be + to label the overall instance with +. Given that any boolean expression can be written in the conjunctive normal form, $Q_1(x^1) \wedge \ldots \wedge Q_K(x^K)$ already covers a large class of decision problems, while our work also supports other decision functions that use $\vee$.

Given the decision function F, we aim to learn the subspatial decision functions, $\{Q_1(x^1), \ldots, Q_K(x^K)\}$, efficiently from a small set of labeled instances. For a labeled instance x, the user provides a collection of *subspatial labels* $(y^1, \ldots, y^K) \in \{+,-\}^K$ to enable learning. We note that while this process requires more annotation effort from the user, it should not demand more *thinking* effort than deriving the global label.

Generality. In what follows, we compare our factorization frameworks with similar works in the literature.

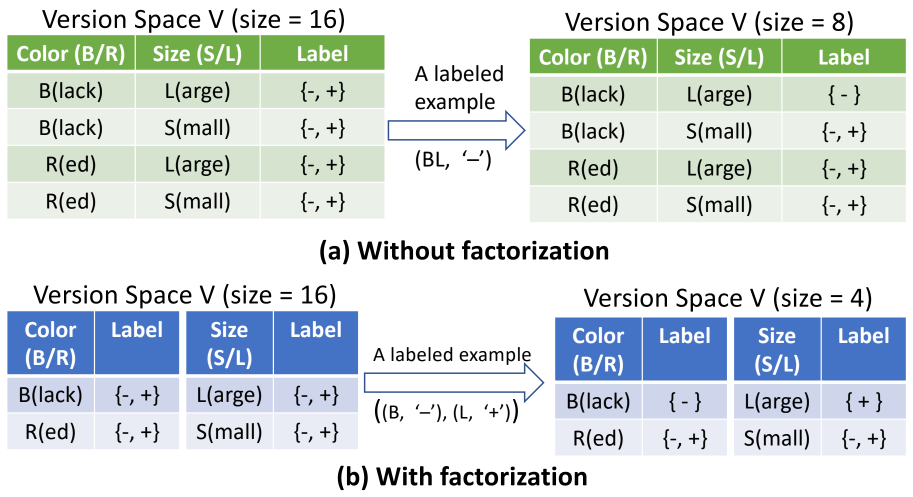
First, we note the differences of our factorization framework from Huang et al. (2018): First, one of the main assumptions in Huang et al. (2018) is that the set $\{x^k : Q_k(x^k) = +\}$ or the set $\{x^k : Q_k(x^k) = -\}$ must be a convex object, which is eliminated in this work. Second, the global decision function F must be conjunctive in Huang et al. (2018), which is relaxed to any boolean expression in our work. Third, our factorization is applied to version space algorithms, as shown below, while Huang et al. (2018) does not consider them at all.

Another line of related work is the data programming systems such as Snorkel (Bach et al., 2017; Ratner et al., 2016). In such systems, users have to provide labeling functions whose predictions are correlated with the labeling task at hand. However, such functions usually depend on one or more constant values which have to be manually tuned by the user, and can have a large impact on its prediction quality. In a more recent work, Snuba (Varma and Ré, 2018), such constants could be automatically tuned by relying on a small pool of hundreds to thousands of labeled points. In contrast, our factorization framework only requires to know which sets of attributes compose each subspace; no initial pool of labeled data or manual tuning of parameters is required.

Finally, we also discuss how our idea of "factorization" compares with other works in Machine Learning. For example, popular types of factorization are the "matrix factorization" technique in recommender systems, and the "factorization of probability distributions" in probabilistic graphical models. However, the general concept of "breaking something complex into several simple parts" still remains in all of these works including ours: in our case, the user prediction function $Q(x)$ is broken down into several simpler predictors $Q_k(x^k)$, which can then be pieced together into the final prediction $Q(x) = F(Q_1(x^1), \ldots, Q_K(x^k))$.

5.1 Introduction to Factorized Version Space

We now give an intuitive description of factorized version space (while we defer a formal description to Section 5.3).


Version Space V (size = 16)
A labeled example
(BL, '–')
Version Space V (size = 8)
(a) Without factorization
Version Space V (size = 16)
A labeled example
((B, '–'), (L, '+'))
Version Space V (size = 4)
(b) With factorization

Color (B/R)	Size (S/L)	Label
B(lack)	L(arge)	{-, +}
B(lack)	S(mall)	{-, +}
R(ed)	L(arge)	{-, +}
R(ed)	S(mall)	{-, +}

Color (B/R)	Size (S/L)	Label
B(lack)	L(arge)	{ - }
B(lack)	S(mall)	{-, +}
R(ed)	L(arge)	{-, +}
R(ed)	S(mall)	{-, +}

Color (B/R)	Label	Size (S/L)	Label
B(lack)	{-, +}	L(arge)	{-, +}
R(ed)	{-, +}	S(mall)	{-, +}

Color (B/R)	Label	Size (S/L)	Label
B(lack)	{ - }	L(arge)	{ + }
R(ed)	{-, +}	S(mall)	{-, +}

Figure 3: Illustration of factorization on the version space.

Without factorization, our problem is to learn a classifier h (e.g., a kernel classifier) on the attribute set $\boldsymbol{A}$ from a labeled data set, $\mathcal{L} = \{(x_i, y_i)\}$, where x_i is a data instance containing values of $\boldsymbol{A}$, and $y_i \in \{+, -\}$. The version space $\mathcal{V}$ includes all possible configurations of h (e.g., all possible bias and weight vector of the kernel classifier) that are consistent with $\mathcal{L}$.

Given a factorization structure $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^K)$, we define K subspaces, where the k^{th} subspace includes all the data instances projected on $\boldsymbol{A}^k$. Then our goal is to learn a classifier h^k for each subspace k from its labeled set, $\mathcal{L}^k = \{(x_i^k, y_i^k)\}$, where y_i^k is the subspatial label for x_i^k. For the classifier h^k, its version space, $\mathcal{V}^k$, includes all possible configurations of h^k that are consistent with $\mathcal{L}^k$. Across all subspaces, we can reconstruct the version space via $\mathcal{V}_f = \mathcal{V}^1 \times \ldots \mathcal{V}^K$. For any unlabeled instance, x, we can use $F(h^1(x^1), \ldots, h^K(x^K))$ to predict a label.

At this point, the reader may wonder what benefit factorization provides in the learning process. We use the following example to show that factorization may enable faster reduction of the version space, hence enabling faster convergence to the correct classification model.

Example. Figure 3 shows an example that the user considers the color and the size of cars, where the color can be black (B) or red (R) and the size can be large (L) or small (S). Therefore, there can be four types of cars corresponding to different color and size combinations. Figure 3(a) shows that without factorization, and in the absence of any user labeled data, the version space contains 16 possible classifiers that correspond to 16 combinations of the $\{-, +\}$ labels assigned to the four types of cars. Once we obtain the '$-$' label for the type BL (color = Black and size = Large), the version space is reduced to 8 classifiers that assign the '$-$' label to BL.

Next consider factorization. In the absence of labeled data, Figure 3(b) shows that the subspace for color includes two types of cars, B and R, and its version space includes 4 possible classifiers that correspond to 4 combinations of the $\{-, +\}$ labels of these two types of cars. Similarly, the subspace for size also includes 4 classifiers. Combining the color and

Algorithm 4 A Factorized Version Space Algorithm

```
Input: data set $\mathcal{X}$, initial labeled set $\mathcal{L}_0$, data sample size $m_1$, per labeling iteration sub-
    sample size $m_2$, version space sample size $M$
 1: $S \leftarrow subsample(\mathcal{X}, m_1)$
 2: $\mathcal{L} = \mathcal{L}_0, \mathcal{U} = S$
 3: while user is still willing do
 4:     $\mathcal{U}' \leftarrow$ subsample($\mathcal{U}, m_2$)
 5:     for each subspace $k$ do
 6:         $\mathcal{U}^k \leftarrow \{x^k,$ for $x \in \mathcal{U}'\}$
 7:         $\mathcal{L}^k \leftarrow \{(x^k, y^k),$ for $(x, y) \in \mathcal{L}\}$
 8:         $\{p^k_{x,+}(x)\}_{x \in \mathcal{U}'} \leftarrow$ compute_cut_probability($\mathcal{L}^k$,$\mathcal{U}^k$, $M$)
 9:     end for
10:     $x^* \leftarrow$ selection_strategy($\mathcal{U}', p^1_{x,+}, \ldots, p^K_{x,+}$)
11:     $y^* \leftarrow$ get_labels_from_user($x^*$)
12:     $\mathcal{L}, \mathcal{U} \leftarrow \mathcal{L} \cup \{(x^*, y^*)\}, \mathcal{U}/\{x^*\}$
13: end while
14: for $k$ from 1 to $K$ do
15:     $MV^k \leftarrow$ train_majority_vote_classifier($\mathcal{L}^k$)
16: end for
17: $h_{final} \leftarrow x \mapsto F(MV^1(x^1), \ldots, MV^K(x^K))$
18: return $h_{final}$
```

size, we have 16 possible classifiers. Once BL is labeled, this time, with two subspatial labels, ((B,−), (L,+)), each subspace has only two classifiers left, yielding 4 remaining classifiers across the two subspaces. As can be seen, with factorization each labeled instance offers more information and hence can lead to faster reduction to the version space.

5.2 Overview of a Factorized Version Space Algorithm

Based on the above insight, we propose a new active learning algorithm, called a *factorized version space algorithm*. It leverages the factorization structure provided by the user to create subspaces, and factorizes the version space accordingly to perform active learning in the subspaces.

Algorithm 4 shows the pseudo-code of our algorithm. It starts by taking a labeled data set (which can be empty), and creating a memory-resident sample from the underlying large data set as an unlabeled pool $\mathcal{U}$ (lines 1-2). This is done for efficiency reasons in order to guarantee the interactive performance of our algorithms, as discussed in Section 3.5. Then it proceeds to an iterative procedure (lines 3-13): In each iteration, it may further subsample the unlabeled pool to obtain $\mathcal{U}'$ to expedite learning (line 4). Then the algorithm considers each subspace (lines 5-9), including the both labeled instances, $(x^k, y^k) \in \mathcal{L}^k$, and unlabeled instances, $x^k \in \mathcal{U}'$, projected to this subspace. The key step is to compute for each unlabeled instance, x, how much its projection x^k can reduce the version space $\mathcal{V}^k$ once its label is acquired (line 8). This step requires efficient sampling of the version space, $\mathcal{V}^k$, as shown in Section 4. Once the above computation for all subspaces completes, then the algorithm chooses the next unlabeled instance that can result in best reduction of the factorized version space, $\mathcal{V}_f = \mathcal{V}^1 \times \ldots \times \mathcal{V}^K$ (line 10). Our work proposes different strategies and proves the optimality of these strategies, as we detail in Sections 5.3-5.5. The selected instance is then presented to the user for labeling and the unlabeled pool is updated. The algorithm then proceeds to the next iteration.

Once the user wishes to stop exploring, a majority vote classifier (Section 3.5.1) is trained for each subspace k (lines 14-16). Finally, given the classifiers, $(MV^1, \ldots, MV^K)$, for the subspaces, the algorithm builds the final classifier $F(MV^1, \ldots, MV^K)$ (line 17), which can then be used to label all points in the entire data set $\mathcal{X}$.

5.3 Bisection Rule over Factorized Version Space

Let's suppose that a factorization structure $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^K)$ is given. For each subspace k, the user labels the projection x^k of x over $\boldsymbol{A}^k$ based on a hypothesis from a hypothesis class $\mathcal{H}^k$ with prior probability distribution π^k. The user then provides a binary label $\{+, -\}$ for each subspace; in other words, for each x a collection of *subspatial labels* $(y^1, \ldots, y^K) \in \mathcal{Y}_f = \{+, -\}^K$ is provided by the user.

Definition 20 (Factorized hypothesis and factorized hypothesis space) *Define a factorized hypothesis as any function $H : \mathcal{X} \to \mathcal{Y}_f$ such that*

$$H(x) = (h^1, \ldots, h^K)(x) = (h^1(x^1), \ldots, h^K(x^K))$$

H belongs to the product space $\mathcal{H}_f = \mathcal{H}^1 \times \ldots \times \mathcal{H}^K$, which we call the factorized hypothesis space.

We assume that the user labels the subspaces independently and consistently within each subspace.

Definition 21 (Factorized version space) *Let $\mathcal{L}$ be the labeled set at any iteration of active learning. We define the factorized version space as*

$$\mathcal{V}_f = \{[H] \in \mathcal{H}_f \,/\sim\, : H(x) = y \text{ for all } (x, y) \in \mathcal{L}\}$$

Additionally, we note that $\mathcal{V}_f$ is equivalent to $\prod_k \mathcal{V}^k = \mathcal{V}^1 \times \ldots \times \mathcal{V}^K$, where

$$\mathcal{V}^k = \{[h] \in \mathcal{H}^k \,/\sim\, : h(x^k) = y^k \text{ for all } (x, y) \in \mathcal{L}\}$$

is the version space at subspace k.

Given the assumption of independent labeling among the subspaces, the prior probability distribution over the factorized hypothesis space is $\pi_f = \pi^1 \times \ldots \times \pi^K$. Now, by applying the bisection rule to $(\mathcal{X}, \mathcal{Y}_f, \mathcal{H}_f, \pi_f)$, we can define we strategy below:

Definition 22 (Factorized greedy selection strategy) *Let $p_{x,y} = \mathbb{P}(H(x) = y \,|\, [H] \in \mathcal{V}_f)$. The factorized greedy strategy is defined as*

$$\arg\max_{x \in \mathcal{U}} 1 - \sum_{y \in \mathcal{Y}_f} p_{x,y}^2 \tag{12}$$

Additionally, by noting that $\min_{H \in \tilde{\mathcal{H}}} \tilde{\pi}(H) = \prod_k \min_{h_k \in \mathcal{H}^k} \pi^k(h_k)$, the following theorem is the direct application of Equation (2):

Theorem 23 (Number of iterations with factorization) *The factorized greedy strategy in* (12) *takes at most*

$$OPT_f \cdot \left(1 + \sum_{k=1}^{K} \ln\left(\frac{1}{\min_{h_k} \pi_k(h_k)}\right)\right)^2$$

iterations in expectation to identify a hypothesis randomly drawn from π_f, where OPT_f is the minimum number of iterations across all strategies that continue until the target hypothesis over all subspaces has been found.

Finally, we derive a simplified computation of the factorization greedy strategy (12):

Theorem 24 *Let $p^k_{x,\pm} = \mathbb{P}(h(x^k) = \pm \,|\, [h] \in \mathcal{V}^k)$, the factorized greedy selection strategy (12) is equivalent to*

$$\arg\max_{x \in \mathcal{U}} 1 - \prod_{k=1}^{K} (1 - 2\, p^k_{x,+} p^k_{x,-}) \tag{13}$$

5.4 Factorized Squared-Loss Strategy

We also propose a variant of the greedy strategy, which we call the *squared-loss strategy*, for selecting the next example for labeling:

$$\arg\max_{x \in \mathcal{U}} \sum_{k=1}^{K} 2 p^k_{x,+} p^k_{x,-} \tag{14}$$

Intuitively, this strategy follows the same intuition as the greedy strategy: find an example x that simultaneously halves all $\mathcal{V}^k$. This strategy also has very similar performance guarantees to the greedy strategy:

Theorem 25 (Factorized squared loss strategy) *The squared loss strategy takes at most*

$$K \cdot OPT_f \cdot \left(1 + \sum_{k=1}^{K} \ln\left(\frac{1}{\min_{h_k} \pi_k(h_k)}\right)\right)^2$$

iterations in expectation.

5.5 Factorized Product-Loss Strategy

One problem with the previous strategies is that they attempt to find $H^* \in \mathcal{H}_f$ which correctly predicts all partial labels, without taking into account the *final prediction* $F(H(x))$ of our classifier. For example, if changing a single partial label does not affect the final prediction $F(H(x))$, it is not worth spending any effort in reducing the version space at this particular subspace.

With this idea in mind, let us define a function $\tilde{F}$ mapping each $H \in \mathcal{H}_f$ into the classifier making the final predictions: $\tilde{F}(H)(x) = F(H(x))$. Given some labeled data $\mathcal{L}$, the set of final, consistent classifiers is given by

$$\tilde{\mathcal{V}}_f = \{\tilde{F}(H) : H(x) = y, \forall x, y \in \mathcal{L}\}$$

Now, our selection strategy follows the same principle of the version space bisection rule: select the point x which halves the set of consistent classifiers $\tilde{\mathcal{V}}_f$ in half, that is, it selects

$$\arg\max_{x \in \mathcal{U}} 2\tilde{p}_{x,+}\tilde{p}_{x,-} \tag{15}$$

where $\tilde{p}_{x,\pm} = \mathbb{P}(F(H(x)) = \pm \,|\, [H] \in \mathcal{V}_f) = \sum_{y:F(y)=\pm} \prod_{k=1}^{K} p^k_{x,y_k}$. We call it the *product-loss* strategy, because in the popular case of conjunctive F it simplifies to

$$\tilde{p}_{x,+} = \prod_{k=1}^{K} p^k_{x,+}$$

We note that the above strategy corresponds to a hybrid between the unfactorized and factorized version space bisection rules. On one hand, it splits the version space according to the binary prediction $F(H(x)) \in \{-,+\}$, repeating this procedure until one single equivalence class $[\tilde{F}(H^*)]$ of binary classifiers remains. On the other hand, the version space $\tilde{\mathcal{V}}_f$ itself is composed of classifiers consistent with all partial labels, and not only the final labels. In conclusion, the hybrid nature of this strategy should intuitively guarantee a faster convergence speed than our previous factorized strategies.

5.6 Optimization for Categorical Subspaces

The factorization framework allows us to treat categorical variables in a more natural way. Assume that in a given subspace all attributes are categorical. Instead of encoding these values to numbers (e.g., using the one-hot encoding), we consider a categorical hypothesis class $\mathcal{H}^{cat} = \{h : \mathcal{C} \to \{-,+\}\}$, where $\mathcal{C}$ is the set of all categories present in the data. Assuming a uniform prior over hypotheses, it is easy to see that the positive cut probabilities are given by

$$p_{c,+} = \begin{cases} 1, & \text{if } (c,+) \in \mathcal{L} \\ 0, & \text{if } (c,-) \in \mathcal{L} \\ 0.5, & \text{otherwise} \end{cases} \tag{16}$$

This can be directly plugged into Equations (13) or (14). As for prediction, we simply use the majority vote classifier C^{cat} definition, $C^{cat}(c) = + \iff p_{c,+} > 0.5 \iff (c,+) \in \mathcal{L}$.

We also note that these cut probabilities are extremely efficient to compute. Compared to our version space algorithm, it does not require any special sampling procedure or optimization: a simple memorization of the categories and labels is enough.

6. Experiments

We implemented all of our techniques in a Python-based prototype.[6] In this section, we evaluate our techniques against state-of-the-art active learning algorithms (Tong and Koller, 2001; Settles, 2012; Gonen et al., 2013; Huang et al., 2018) in terms of accuracy (using F-score) and efficiency (execution time in each labeling iteration). Note that F-score is the harmonic mean of precision and recall for retrieving objects in the positive class, and a suitable measure when the user interest covers a small portion of the data set, which is often the case in large data set exploration. Our evaluation particularly focuses on F-score in the first 200 labeling iterations as our work aims to address the cold start problem.

6.1 Experimental Setup

Data sets and user interest patterns: We evaluate our techniques using two real-world data sets and user interest patterns.

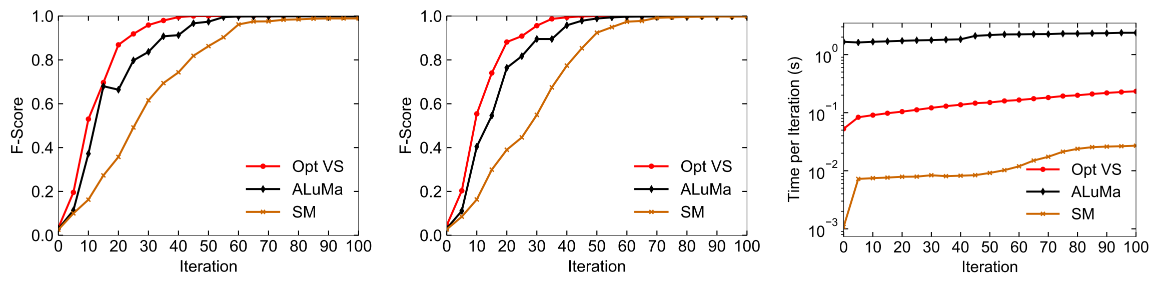
(1) Sloan Digital Sky Survey (SDSS, 190 million points): The data set contains the "PhotoObjAll" table with 510 attributes and 190 million sky observations.[7] We used a 1% sample (1.9 million points, 4.9GB) to create a data set for running active learning algorithms—our formal result in Section 3.5 allowed us to use a sample for efficient execution, yet being able to approximate the accuracy over the entire data set.

SDSS also offers a query release where the SQL queries reflect the data interest of scientists.[8] We extracted 11 queries to build a benchmark, as shown in Appendix D, and treated them as the ground truth of the positive classes of 11 classifiers to be learned—our system does *not* need to know these queries in advance, but can learn them via active learning. These queries reflect different dimensionality (2D-7D) and complexity of the decision boundary (e.g., various combinations of linear, quadratic, log patterns). As their decision boundaries all lie in dense data regions, these queries require large numbers of labeled instances to learn the boundaries precisely.

(2) Car data set (5622 points): This small data set was used in Huang et al. (2018) to conduct a user study, which generated 18 queries representing the true user interests. We obtained this data set and queries from the authors of Huang et al. (2018). These queries include 4-10 attributes, with a mix of both numerical and categorical attributes, and have selectivities in [0.2%, 0.9%]. Categorical attributes were preprocessed using one-hot-encoding, resulting in 4-418 features. Since the data is sparse in this data set, it is easier to learn the decision boundaries than in SDSS.

Algorithms: We compare our algorithms to state-of-the-art VS algorithms, including Simple Margin (SM) (Tong and Koller, 2001), Query-by-Disagreement (QBD) (Settles, 2012), and ALuMA (Gonen et al., 2013), as well as a factorization-aware algorithm, DSM (Huang et al., 2018). When selecting the next point to label, each algorithm only considers a sample of at most 50,000 unlabeled points. In our experiments, each active learning algorithm starts with one positive example and one negative example chosen at random, and runs up

6. Our code is available at `https://gitlab.inria.fr/ldipalma/aideme`
7. `http://www.sdss3.org/dr8/`
8. `http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp`



(a) 10000 data points, F-score (b) 40000 data points, F-score (c) 40000 data points, Time

Figure 4: Comparison of OptVS, ALuMa, and Simple Margin (SM) over a synthetic data set of various sizes.

to 200 additional labeled examples (iterations). Each experiment was repeated 10 times and individual results were averaged.

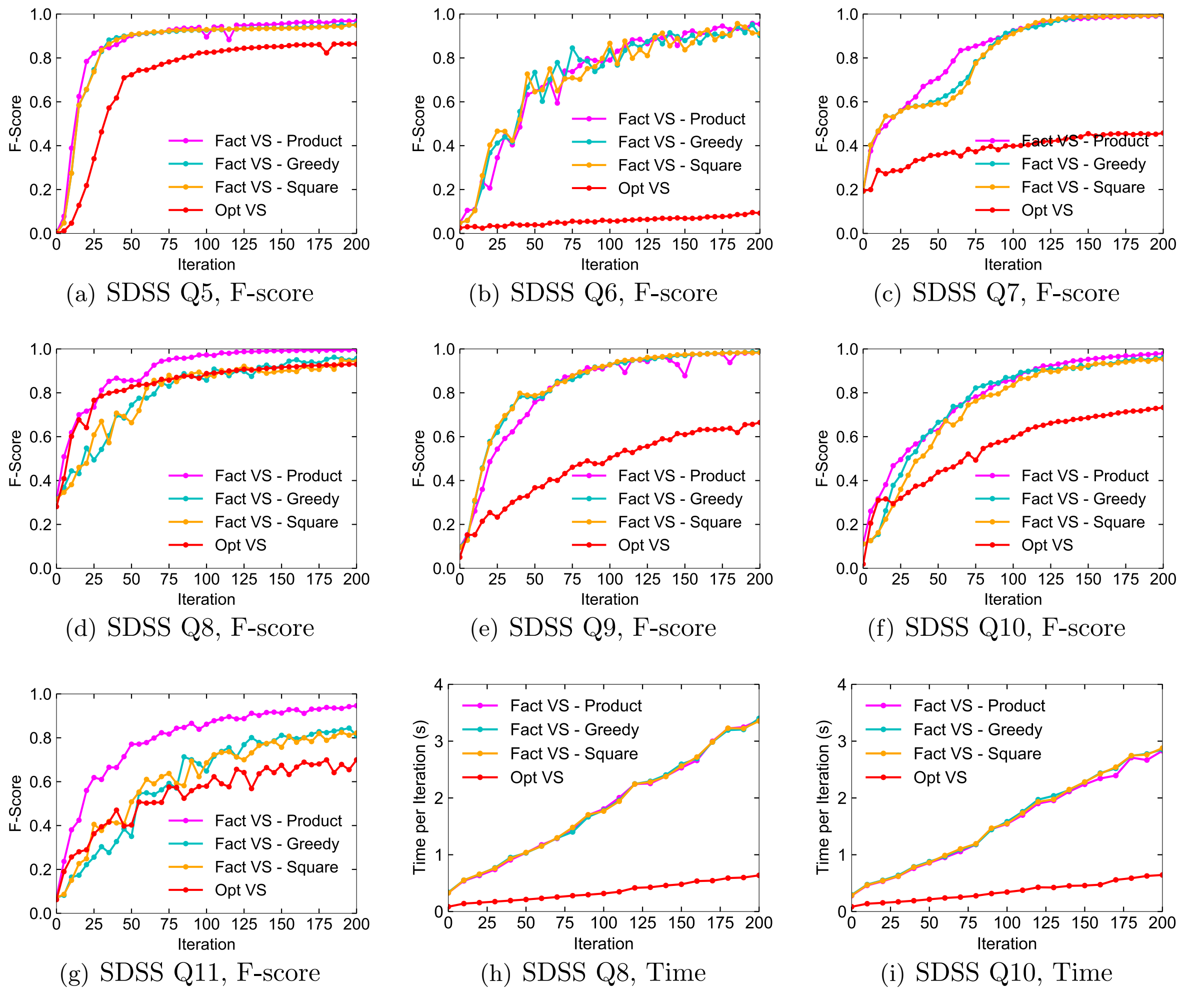
Hyper-parameter tuning: Hyper-parameters were tuned to achieve the highest accuracy while running under interactive performance (or close to). All active learners use a RBF kernel with a default scaling of 1/num_features, except for the SDSS Q4 experiment for which we chose a scaling of 0.001. Algorithms relying on SVM (SM, QBD, DSM) use a penalty of $C = 10^5$. For QBD, we use a background sample of 200 points which are weighted by 10^{-5} during training. As for ALuMA, it samples 16 hypotheses from the version space, each one generated from an independent hit-and-run chain of length of 2000. Additionally, our OptVS algorithm also samples 16 hypotheses, but they are generated from a single hit-and-run chain with warm-up and thin parameters equal to 100 steps. In OptVS, we also added a small jitter of 10^{-12} to the diagonal of the kernel matrix, making it positive-definite. Additionally, we chose a rounding parameter $\gamma = 1/(n+1)\sqrt{n}$, where n is the dimension of W_t. The factorized version space algorithm uses the same parameters for sampling each version subspace. Finally, all factorized algorithms assume a conjunctive labeling function.

Servers: Our experiments were run on four servers, each with 40-core Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 128GB memory, Python 3.7 on CentOS 7.

6.2 Comparison with ALuMA

One major limitation of the ALuMA (Gonen et al., 2013) algorithm is its costly preprocessing step: in order to support kernels, it requires computing the kernel matrix over the entire data set before applying a dimensionality reduction technique, a process that is very memory intensive. For this reason, we could not run this preprocessing for the SDSS data set of 1.9M data points, since a $10^6 \times 10^6$ matrix would consume terabytes of memory.

Expt 1 (Comparison with ALuMA): In order to enable a comparison with ALuMA, we have generated small synthetic data sets composed of $N \in [1 \times 10^4, 4 \times 10^4]$ data points sampled uniformly over $[-\sqrt{3}, \sqrt{3}]^2$; we could not run experiments for $N > 40k$ due to memory constraints of our servers. The user interest region is a circle centered at the origin with radius chosen to guarantee a selectivity of 1%, similar to the SDSS and car queries.



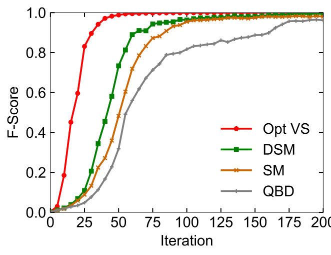
(a) SDSS Q5, F-score
(b) SDSS Q6, F-score
(c) SDSS Q7, F-score
(d) SDSS Q8, F-score
(e) SDSS Q9, F-score
(f) SDSS Q10, F-score
(g) SDSS Q11, F-score
(h) SDSS Q8, Time
(i) SDSS Q10, Time

Figure 5: Effects of factorization on performance using SDSS data set

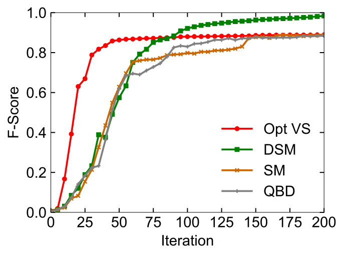
After applying the ALuMA preprocessing, the transformed data set has a dimensionality close to 300. We note that while ALuMA is run over this transformed data set, all other algorithms will be run over the original data set.

The comparative results are shown in Figure 4. First, in terms of accuracy our OptVS algorithm tends to outperform ALuMA in every iteration, which in turn outperforms other VS algorithms. With respect to the time measurements, ALuMA is by far the most expensive, being around 10 times slower than OptVS. On the other hand, Simple Margin runs more quickly than OptVS since it is a rough approximation of the bisection rule; however, its improved running time comes with a penalty in accuracy, performing the worst among all algorithms considered.

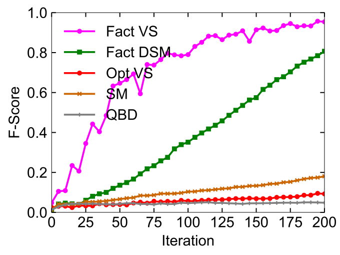
As we can see from these experiments, although ALuMA can provide a reasonable performance in terms of convergence speed, its preprocessing step is too costly to run for large data sets and, even if possible, it is still does not match our OptVS algorithm in either



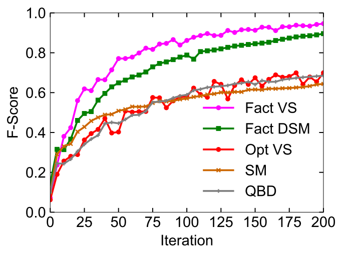
(a) Opt VS versus other algs (Q2, F-score)



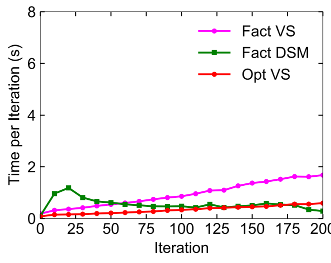
(b) Opt VS versus other algs (Q3, F-score)



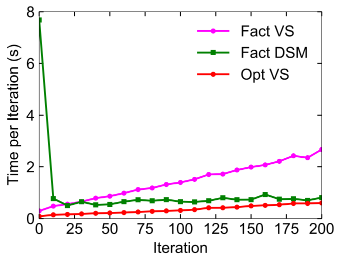
(c) Fact VS versus other algs (Q6, F-score)



(d) Fact VS versus other algs (Q11, F-score)



(e) Fact VS versus other algs (Q6, time)



(f) Fact VS versus other algs (Q11, time)

Figure 6: Comparison to active learning algorithms (Tong and Koller, 2001; Settles, 2012; Huang et al., 2018) using SDSS

accuracy or processing time. Thus, in the SDSS and Car data set experiments we will leave out ALuMA from further consideration.

6.3 Evaluating Our Techniques Using SDSS

Expt 2 (Factorization): We next study the effect of factorization, by comparing Opt VS with its extension to factorization, denoted as “Fact VS”. For factorization, each predicate in the target query corresponds to its own factorized subspace. Q1-Q4 are not factorized because they are 2D queries. Q5-Q7 are factorized with each predicate corresponding to each subspace with no overlap of attributes across subspaces, while Q8-Q11 are also factorized but with overlap of attributes across subspaces.

For Q5-Q11, Fact VS outperforms Opt VS, often by a wide margin. Figures 5(a)-5(g) show the F-score for Q5-Q11. In all cases, factorization can provide a significant boost in performance; for Q6 in particular, the factorized version reaches $> 90\%$ F-Score after 200 iterations while the non-factorized version is still at less $< 10\%$. In addition, we do not observe any performance difference between the greedy and squared loss strategies of factorization, despite the latter having a worse theoretical bound. As for the product loss, results are nearly identical to the other two losses for most queries, except Q8 and Q11 for which the product loss performs significantly better. This confirms our intuition of Section 5.5 that the product loss should converge faster since it attempts to directly reduce the number of final classifiers, instead of the factorized ones.

Query #	01			02			03			04			05			06		
Fact VS	–	–	–	–	–	–	–	–	–	–	–	–	**82**	**90**	**90**	**35**	**65**	**79**
DSM	10	59	**93**	11	73	96	19	57	**92**	26	80	98	6	43	87	6	14	35
Opt VS	**81**	**90**	90	**83**	**99**	**100**	**67**	**86**	88	**97**	**100**	**100**	34	72	82	3	4	6
SM	8	42	90	9	48	95	15	63	80	86	98	**100**	3	37	74	5	7	10
QBD	4	26	81	5	32	82	18	62	83	91	96	97	0	2	24	4	4	5

Query #	07			08			09			10			11		
Fact VS	56	**71**	**92**	73	86	**97**	**54**	**76**	**93**	50	63	86	**62**	**77**	**86**
DSM	**62**	70	65	76	**87**	94	43	68	87	**55**	**76**	**89**	50	65	79
Opt VS	29	36	40	**77**	83	89	23	37	50	32	44	60	36	40	58
SM	44	48	51	70	81	86	32	37	47	34	50	59	43	51	57
QBD	41	46	54	64	78	86	27	35	47	29	39	53	34	45	59

Table 1: F-score (in %) comparison at iterations 25, 50, and 100 for SDSS data set queries.

Finally, the running time of Fact VS is somewhat higher than Opt VS, due to the need to handle multiple subspaces and our current implementation that runs subspatial computations serially. Figures 5(h) and 5(i) shows the time for Q8 and Q10, which are the two most expensive queries with at least 5 subspaces each. The time per iteration is always below 4 seconds, and hence satisfies the requirement of interactive speed. Note that the time per iteration can be further improved if subspatial computations are run concurrently, which is left to our future work.

6.4 Comparing to Other Techniques Using SDSS

Expt 3 (Comparison to other algorithms): We next compare our algorithms to two VS algorithms, Simple Margin (SM) (Tong and Koller, 2001) and Query-by-Disagreement (QBD) (Settles, 2012), as well as DSM (Huang et al., 2018), a factorization-aware algorithm. Our Factorized Version Space algorithm uses the product loss, which tends to give better results. Figure 6 illustrates per-iteration measurements for a subset of queries, while Table 1 shows a complete set of results for all 11 queries.

We first consider the case without factorization. Our OptVS algorithm outperforms the two VS algorithms (Tong and Koller, 2001; Settles, 2012) and DSM (Huang et al., 2018) most time for the low-dimensional queries, Q1-Q4, when factorization does not apply. Figures 6(a) and 6(b) show the results for Q2 and Q3. During the initial iterations, OptVS improves much faster than other algorithms, and remains the best across all iterations of Q2 and in most iterations of Q3. For Q3, the improvement of OptVS slows down in later iterations, due to the use of a Gaussian kernel as our default kernel. Since Q3 is a rectangle pattern, the decision boundary of the Gaussian kernel tends to be round in shape, making it hard to fit a rectangle pattern. In contrast, DSM builds a polytope model that can more easily capture a rectangle pattern.

We next consider factorization. Our Fact VS almost always outperforms others, including DSM that uses factorization and stronger assumptions. Figures 6(c) and 6(d) show the results for Q6 and Q11. In general, Fact VS outperforms DSM, which in turn is an upper

Query #	01			02			03			04			05			06		
Fact VS	**76**	90	**100**	**96**	**100**	**100**	**67**	**86**	**100**	**65**	**90**	**100**	**100**	**100**	**100**	**84**	**100**	**100**
DSM	51	**94**	**100**	82	**100**	**100**	21	54	**100**	56	82	**100**	68	83	**100**	78	**100**	**100**
Opt VS	36	57	73	29	62	89	19	49	89	8	28	51	43	49	83	54	64	80
SM	27	43	43	20	26	70	13	14	47	5	12	35	43	43	43	44	77	**100**
QBD	29	29	29	10	12	14	11	11	12	4	5	6	43	43	43	25	28	29
Query #	07			08			09			10			11			12		
Fact VS	**75**	89	**100**	**100**	**100**	**100**	89	**100**	**100**	**69**	**96**	**100**	**69**	**97**	**100**	**71**	**98**	**100**
DSM	**75**	**95**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	61	87	**100**	54	89	**100**	53	93	**100**
Opt VS	42	70	79	95	**100**	**100**	59	71	**100**	19	38	63	10	29	61	16	51	80
SM	45	56	71	53	**100**	**100**	46	71	97	9	16	34	5	15	43	6	16	58
QBD	28	32	36	28	34	39	23	25	41	7	9	10	4	4	5	4	6	9
Query #	13			14			15			16			17			18		
Fact VS	**90**	**99**	**100**	**94**	**100**	**100**	**65**	**87**	**100**	**99**	**100**	**100**	**35**	56	**100**	**98**	**100**	**100**
DSM	79	**99**	**100**	72	98	**100**	60	**87**	**100**	98	**100**	**100**	**35**	**63**	**100**	**98**	**100**	**100**
Opt VS	25	41	58	37	59	92	27	36	62	54	75	99	17	29	40	66	95	96
SM	18	32	40	29	37	65	21	37	45	50	84	**100**	12	16	32	72	91	**100**
QBD	9	10	13	21	21	21	21	23	22	15	24	31	7	7	7	41	48	62

Table 2: F-score (in %) comparison at iterations 5, 10, and 20 for Cars data set queries.

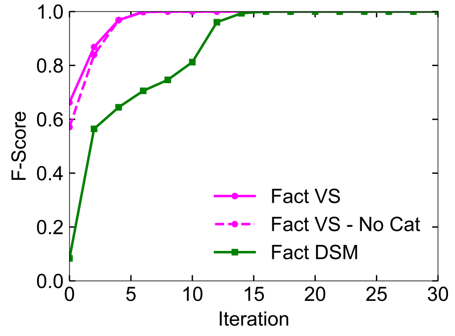
bound in accuracy of all other (non-factorized) algorithms. In the particular case of 6(c), after 100 iterations Fact VS is at $> 80\%$ accuracy, while DSM is still at 40% and all of the other alternatives are at 10% or lower.

Finally, Figures 6(e) and 6(f) shows the running time of Fact VS, DSM, and OptVS for Q6 and Q11. In both cases, the two factorized algorithms take at most a couple of seconds per iteration, thus being compatible in the interactive data exploration scenario. In addition, we notice that DSM has a large warm-up time, being slower than Fact VS during the first 30 iterations. Thus, Fact VS may be preferred to DSM given its better accuracy and lower time per iteration in initial iterations.

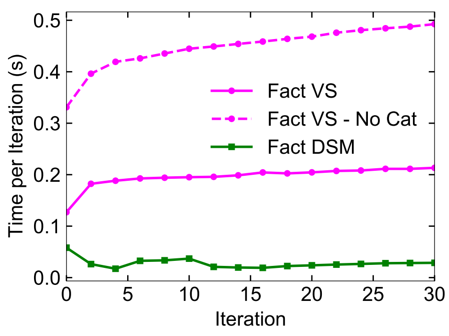
6.5 Comparing to Other Techniques Using Car Queries

For the 18 queries over the cars data set, we do not observe a big performance difference between our factorized VS algorithm (with product loss) and DSM. In all cases, both algorithms reach 100% accuracy in less than 20 iterations. This fast convergence is due to sparsity of data, thus allowing more freedom in how we separate the positive and negative classes. Refer to Table 2 for a more complete set of results.

Expt 4 (Optimization for categorical attributes): We next study the effect of our optimization for categorical variables, as described in Section 5.6. Figure 7 shows the result for one example query. The "No Cat" version of our algorithm relies on the one-hot-encoding of the data. As we can see, this optimization does not have a visible impact on accuracy, but it can significantly reduce the time per iteration, providing a speed-up of around 100% in iteration time. We also observed similar results over all other car queries.



(a) Car Q5, F-score



(b) Cars Q5, Time

Figure 7: Optimization of categorical variables using the car data set

6.6 Summary of Experimental Results

From the previous experiments, we can draw the following conclusions regarding our proposed algorithms.

First, we have observed that OptVS outperforms other VS algorithms in the vast majority of experiments. In particular, we note that OptVS can reach high accuracy with a small number of labeled points: in SDSS Q1-Q3, OptVS reached an accuracy larger than 67% after only 25 labeling iterations, while all other VS algorithms were still below 19% F-score. In addition, OptVS was also shown to outperform DSM in non-factorized scenarios. In terms of time measurements, OptVS runs under 1 second per iteration at all times, being around 10 times faster than ALuMA.

Second, FactVS was shown to outperform OptVS in almost all high-dimensional cases, irrespective of the loss being used. In addition, we did not observe any performance difference between the Greedy and the Squared losses, but we did observe a few cases where the Product loss significantly outperforms the other two.

Third, for higher dimensional problems, our FactVS algorithm significantly outperforms other VS algorithms, as well as DSM, a factorization-aware algorithm, for complex tasks while maintaining interactive speed. For example, for a complex user interest pattern tested, our algorithm achieves F-score of over 80% after 100 iterations of labeling, while DSM is still at 40% and all other VS algorithms are at 10% or lower. The difference between FactVS and DSM is reduced for simpler data sets such as the Cars data set due to the sparsity of the data set and the resulting fast convergence. In terms of time measurements, FactVS tends to be faster than DSM in initial labeling iterations, while still running under 4 seconds per iteration at all times.

Finally, our optimization for categorical subspaces was shown to significantly improve the running time of our algorithms, without compromising the accuracy.

7. Conclusion

To overcome the slow convergence problem of active learning (AL) for model development over large datasets, we presented three major contributions: (1) a novel theoretical framework for version space based AL algorithms over kernel classifiers, with strong theoretical guarantees on performance and optimizations for large data sets; (2) an efficient algorithm implementing our proposed strategy, introducing several optimizations for sampling the version space; and (3) an extended version space algorithm that is based on the factorization structure employed in the user labeling process, factorizes the version space to perform active learning in a set of subspaces. Using real world data sets, our evaluation results show that our algorithms significantly outperform state-of-the-art version space algorithms including Simple Margin (Tong and Koller, 2001), Query-by-Disagreement (Settles, 2012), and ALuMA (Gonen et al., 2013), as well as a factorization-aware algorithm, DSM (Huang et al., 2018), while maintaining interactive speed.

For future work, one direction is to extend our version space based techniques to handle noisy labels, which are likely to occur in real applications. Another line of work is to extend the factorization technique to other classes of Active Learning algorithms, such as Uncertainty Sampling or Expected Error Reduction methods. Furthermore, we will explore how the interactive data exploration framework could be integrated with Data Programming (Ratner et al., 2016; Bach et al., 2017) for efficient model development, combining our fast convergence in early iterations of human labeling with the auto-labeling functionality of Data Programming to scale to large datasets.

Appendix A. Proofs

Below we outline the proofs of our main theoretical results.

A.1 Proof of Theorem 5

This theorem is a simple consequence of the near-optimality result of equation (2) for the general GBS strategy. In fact, it is just a matter of noticing that

$$\begin{aligned} p_{x,y} &= \mathbb{P}_{[h]\sim\pi}(h(x) = y \mid [h] \in \mathcal{V}) \\ &= \mathbb{P}_\theta\left(\psi^{-1}(\{h : h(x) = y\}) \mid \psi^{-1}(\{h : [h] \in \mathcal{V}\})\right) \\ &= \mathbb{P}_\theta\left(\{\theta : h_\theta(x) = y\} \mid \{\theta : [h_\theta] \in \mathcal{V}\}\right) \\ &= \mathbb{P}_\theta(h_\theta(x) = y \mid \theta \in \mathcal{V}_\theta) \end{aligned}$$

which concludes the proof. ■

A.2 Proof of Theorem 6

Since $S = span(\phi(x_1), \dots, \phi(x_N))$ is a finite-dimension subspace of the Hilbert Space $\mathcal{F}$, we can write $\mathcal{F} = S \oplus S^\perp$, where $S^\perp$ is the orthogonal subspace of S. In particular, this guarantees that any $f \in \mathcal{F}$ can be written as

$$f = f|_S + f|_{S^\perp}$$

Thus, for any $x_i \in \mathcal{X}$ we have

$$\langle \phi(x_i), f\rangle_\mathcal{F} = \langle \phi(x_i), f|_S\rangle_\mathcal{F} + \langle \phi(x_i), f|_{S^\perp}\rangle_\mathcal{F} = \langle \phi(x_i), f|_S\rangle_\mathcal{F}$$

In conclusion, we also note that

$$h_{b,f}(x_i) = \text{sign } b + \langle \phi(x_i), f\rangle_\mathcal{F} = \text{sign } b + \langle \phi(x_i), f|_S\rangle_\mathcal{F} = h_{b,f|_S}(x_i)$$

which finishes our proof. ■

A.3 Proof of Lemma 8

The proof of this result relies on the following lemma, which is an extension of Theorem 1 from Gilad-Bachrach et al. (2006):

Lemma 26 *Let $W = \{w \in \mathbb{R}^N : \|w\| \leq 1$ and $a_i^T w \geq 0, 1 \leq i \leq t\}$ and some $a^* \in \mathbb{R}^N$. Then, for any linear subspace S containing $span(a_1, \dots, a_t, a^*)$, we have*

$$\mathbb{P}_{w\sim Unif(W)}(\pm w^T a^* > 0) = \mathbb{P}_{w\sim Unif(W\cap S)}(\pm w^T a^* > 0)$$

Proof Let R be any rotation mapping S into $\mathbb{R}^k \times \{0\}^{N-k}$, where $k = dim(S)$. Since rotations preserve norms and scalar products, we have that

- W is mapped into $\tilde{W} = RW = \{\tilde{w} \in \mathbb{R}^N : \|\tilde{w}\| \leq 1$ and $\tilde{a}_i^T \tilde{w} \geq 0, 1 \leq i \leq t\}$, where $\tilde{w} = Rw$ and $\tilde{a}_i = Ra_i$

- $w^T a^* = \tilde{w}^T \tilde{a}^*$, with $\tilde{a}^* = Ra^*$
- The random variable $w \sim Unif(W)$ is mapped into $\tilde{w} = Rw \sim Unif(\tilde{W})$

From these observations, it is clear that if the lemma holds true for the subspace $\mathbb{R}^k \times \{0\}^{N-k}$, then it must also hold true for S. In what follows, we restrict ourselves to this particular case.

Now, let $U \in \mathbb{R}^k$ be uniformly distributed over $W \cap S$. Its p.d.f. is given by

$$p_U(u) \propto \mathbb{1}(u \in W \cap S) = \mathbb{1}(\|u\| \leq 1)\mathbb{1}(\tilde{a}_i^T u \geq 0, \forall i)$$

where $\tilde{a_i} = (a_i^1, \dots, a_i^k)$. Now, let's consider the random vector $V \in \mathbb{R}^{N-k}$ which is sampled conditionally on $U = u$ according to the distribution

$$p_{V|U=u}(v) \propto \mathbb{1}\left(\|u\|^2 + \|v\|^2 \leq 1\right)$$

Finally, let's define the random vector $\omega = (U_1, \dots, U_k, V_1, \dots, V_{N-k}) \in \mathbb{R}^N$. This vector is distributed according to

$$p_\omega(w) = p_U(w_1, \dots, w_k) P_{V|U=(w_1,\dots,w_k)}(w_{k+1}, \dots, w_N) \propto \mathbb{1}\left(\|w\|^2 \leq 1\right) \mathbb{1}(a_i^T w \geq 0, \forall i)$$

from where we can conclude that $\omega \sim Unif(W)$. With this, using the fact that $a^* \in S = \mathbb{R}^k \times \{0\}^{N-k}$ we can finally conclude that

$$\mathbb{P}_{w \sim Unif(W)}(\pm w^T a^* > 0) = \mathbb{P}_{U \sim Unif(W \cap S)}(\pm U^T \tilde{a}^* > 0)$$

which concludes our demonstration. ■

With the above lemma, we can finally prove our main result. Let's define $a_i = y_i(1, L_i)$ for $1 \leq i \leq t$, $a^* = (1, L_j)$, and $w = (b, \beta)$. Then, we have that:

- $\mathcal{V}_\beta = \{w : \|w\| \leq 1 \text{ and } a_i^T w \geq 0,\ 1 \leq i \leq t\}$, which is identical to W
- $S_j = \mathbb{R} \times span(L_1, \dots, L_t, L_j)$ clearly contains $span(a_1, \dots, a_t, a^*)$

With this, we can apply our lemma to $\mathcal{V}_\beta$ and S_j, which implies that

$$p_{x_j,\pm} = \mathbb{P}_{w \sim Unif(\mathcal{V}_\beta)}(\pm a^* w > 0) = \mathbb{P}_{w' \sim Unif(\mathcal{V}_\beta \cap S_j)}(\pm a^* w' > 0) = \mathbb{P}(\pm(b' + L_j^T \beta') > 0)$$

where $w' = (b', \beta')$ is drawn uniformly over $\mathcal{V}_\beta \cap S_j$. ■

A.4 Proof of Lemma 9

Let's define $\tilde{L}_i = (0, L_i^1, \dots, L_i^N)$. We first observe that

$$S_j = \mathbb{R} \times span(L_1, \dots, L_t, L_j) = span(e_1, \tilde{L}_1, \dots, \tilde{L}_t, \tilde{L}_j)$$

Now, by using the fact $L_i^k = 0$ for $k > i$ and $L_i^i \neq 0$ for all i, we can see that

$$span(\tilde{L}_1, \dots, \tilde{L}_t) = \{0\} \times \mathbb{R}^t \times \{0\}^{N-t-1} = span(e_2, \dots, e_{t+1})$$

which implies $S_j = span(e_1, \dots, e_{t+1}, \tilde{L}_j)$. Finally, the last orthogonal basis vector can constructed by projecting $\tilde{L}_j$ onto $span(e_1, \dots, e_{t+1})^\perp$:

$$T_j = proj_{span(e_1,\dots,e_{t+1})^\perp} \tilde{L}_j = (\vec{0}_{t+1}, L_j^{t+1}, \dots, L_j^N)$$

which concludes our demonstration. ■

A.5 Proof of Theorem 11

By Lemma 8, the cut probabilities satisfy $p_{x_j,+} = \mathbb{P}(b' + \beta'^T L_j > 0)$, where (b', β') is uniformly distributed from $\mathcal{V}_\beta \cap S_j$. Now, by replacing Equation (6) into the definition for $\mathcal{V}_\beta$, we obtain

$$\begin{aligned}
&(b', \beta') \in \mathcal{V}_\beta \cap S_j \\
&\quad \iff (b', \beta') = \sum_{i=1}^{t+1} w_i e_i + w_{t+2} \frac{T_j}{\|T_j\|} \text{ and } b'^2 + \|\beta'\|^2 \le 1 \text{ and } y_i(b' + L_i^T \beta') > 0,\ 1 \le i \le t \\
&\quad \iff \|w\|^2 \le 1 \text{ and } y_i \left(w_1 + \sum_{k=2}^{t+1} L_i^k w_k \right) > 0,\ 1 \le i \le t \\
&\quad \iff \|w\|^2 \le 1 \text{ and } y_i \ell_i^T w > 0,\ 1 \le i \le t \\
&\quad \iff w \in W_t
\end{aligned}$$

In particular, this implies that sampling $(b', \beta') \sim Unif(S_j \cap V_\beta)$ is equivalent to sampling $w \sim Unif(W_t)$. Finally, if we replace the Equation (6) into the cut probability equality above, we obtain

$$\begin{aligned}
p_{x_j,+} &= \mathbb{P}(b' + \beta'^T L_j > 0) \\
&= \mathbb{P}\left(w_1 + \sum_{i=2}^{t+1} w_i e_i^T L_j + w_{t+2} \frac{T_j^T L_j}{\|T_j\|} > 0 \right) \\
&= \mathbb{P}\left(w_1 + \sum_{i=2}^{t+1} L_j^i w_i + \|T_j\| w_{t+2} > 0 \right) \\
&= \mathbb{P}(\ell_j^T w > 0)
\end{aligned}$$

With this, our proof is concluded by simply noticing that

$$\|T_j\| = \sqrt{\|L_j\|^2 - \sum_{k=1}^{t} (L_j^k)^2} = \sqrt{K_{jj} - \sum_{k=1}^{t} (L_j^k)^2}$$

■

A.6 Proof of Lemma 13

Our objective is to apply Theorem 2.2 from Mohri et al. (2012). Let $p(x)$ be any prior probability over a data space $\mathcal{D}$, let S be an i.i.d sample from $p(x)$ of size m, and let $\mathcal{H}'$

be a finite set of hypothesis mapping $\mathcal{D}$ into the label set $\mathcal{Y}$. Then, for any $\delta \in (0, 1)$, with probability $1 - \delta$ it holds that

$$\forall h \in \mathcal{H}', \, \mathbb{P}_X(h(X) \neq h^*(X)) \leq \epsilon_{acc}(h, S) + \sqrt{\frac{\log |\mathcal{H}'| + \log \frac{2}{\delta}}{2m}}$$

With this, our result easily follows by considering two points:

1. First, we choose $\mathcal{D} = \mathcal{X}$ and we select the prior $p(x_i) = \frac{1}{N}$. In particular, S must be a sample with replacement from $\mathcal{X}$, and $\mathbb{P}_X(h(X) \neq Y) = \epsilon_{acc}(h, \mathcal{X})$
2. We choose $\mathcal{H}' = \mathcal{H} \, / \sim$. This is possible because every $[h] \in \mathcal{H} \, / \sim$ can be considered as a classifier $\mathcal{X} \to \mathcal{Y}$ defined by $[h](x_i) = h(x_i), \forall i$; in addition, $\hat{\mathcal{H}}$ is finite.

Our result is finally proved by noticing that $\epsilon_{acc}(h, \cdot)$ is constant over each equivalence class $[h]$. ■

A.7 Proof of Theorem 15

For notation convenience, we will ignore the second parameter in the error functions. The first inequality is easy to prove:

$$\begin{aligned}
\epsilon_{fscore}(h) &= 1 - fscore(h) \\
&= 1 - harmonic\_mean(precision(h), recall(h)) \\
&\leq 1 - \min(precision(h), recall(h)) \\
&= \max(1 - precision(h), 1 - recall(h)) \\
&= \max(\epsilon_{prec}(h), \epsilon_{rec}(h))
\end{aligned}$$

The second inequality can be proved in two parts. First, Theorem 1 from Juba and Le (2019) tells us that $\max(\epsilon_{prec}(h), \epsilon_{rec}(h)) \leq \frac{1}{\mu}\epsilon_{acc}(h)$ whenever $\epsilon_{prec}(h) \leq 0.5$. Second, we have that:

$$\begin{aligned}
\epsilon_{prec} \geq 0.5 &\iff \frac{C^{+,-}}{C^{+,+} + C^{+,-}} \geq 0.5 \\
&\iff C^{+,-} \geq C^{+,+} \\
&\iff C^{+,-} + C^{-,+} \geq C^{+,+} + C^{-,+} \\
&\iff \epsilon_{acc} \geq \mu
\end{aligned}$$

In other words, if $\epsilon_{prec}(h) \geq 0.5$, then $\max(\epsilon_{prec}(h), \epsilon_{rec}(h)) \leq 1 \leq \epsilon_{acc}(h)/\mu$, which concludes our proof. ■

A.8 Proof of Theorem 17

Let X be a random variable sampled uniformly at random from S. Our result is then a simple application of the Markov inequality (Equation C.13 from Mohri et al. 2012):

$$\begin{aligned}\epsilon_{acc}(MV, S) &= \mathbb{P}_X\left(MV(X) \neq h^*(X)\right)\\ &\leq \mathbb{P}_X\left(p_{X,h^*(X)} < 0.5\right)\\ &\leq 2\mathbb{E}_X\left[1 - p_{X,h^*(X)}\right]\\ &= 2\mathbb{E}_X\mathbb{P}_{[H]\sim\pi}(H(X) \neq h^*(X))\\ &= 2\mathbb{E}_{[H]\sim\pi}\mathbb{P}_X(H(X) \neq h^*(X))\\ &= 2\mathbb{E}_{[H]\sim\pi}[\epsilon_{acc}(H, S)]\end{aligned}$$

which concludes our proof. ■

A.9 Proof of Lemma 18

Let's consider an ellipsoid on the form

$$z' = \begin{bmatrix} z \\ 0 \end{bmatrix} \qquad P' = \begin{bmatrix} \lambda P & 0 \\ 0 & \nu \end{bmatrix}$$

where $\lambda, \nu > 0$ to ensure P' is positive-definite. In particular, if we assume $1/\lambda + 1/\nu \leq 1$, $\mathcal{E}'$ can be shown to contain $W_t \times [-1, 1]$:

$$\begin{aligned} w' = (w, w_{t+3}) \in W_t \times [-1, 1] &\Rightarrow w \in W_t \ \wedge\ |w_{t+3}| \leq 1\\ &\Rightarrow w \in \mathcal{E}(z, P) \ \wedge\ |w_{t+3}| \leq 1\\ &\Rightarrow (w - z)^T P^{-1}(w - z) \leq 1 \ \wedge\ |w_{t+3}| \leq 1\\ &\Rightarrow \frac{1}{\lambda}(w - z)^T P^{-1}(w - z) + \frac{1}{\nu} w_{t+3}^2 \leq 1\\ &\Rightarrow w' \in \mathcal{E}(z', P')\end{aligned}$$

as desired. Finally, since we are interested in $\mathcal{E}'$ of volume as small as possible, and $vol(\mathcal{E}') \propto \sqrt{\lambda^{t+2}\nu}$, we obtain the following minimization problem:

$$\text{minimize } \lambda^{t+2}\nu, \ \text{ s.t. } \lambda, \nu > 0 \text{ and } \frac{1}{\lambda} + \frac{1}{\nu} \leq 1$$

This problem can be solved analytically by elementary means, giving as result

$$\lambda^* = 1 + \frac{1}{t+2}, \quad \nu^* = t + 3$$

which concludes our proof. ■

A.10 Proof of Lemma 19

We begin by noticing that W can be written as an intersection of half-spaces:

$$W = \left(\bigcap_{1 \leq i \leq m} H(a_i, 0) \right) \cap \left(\bigcap_{\|w\|=1} H(w, 1) \right)$$

Thus, checking if $\gamma\mathcal{E} \subset W$ can be reduced to the problem of verifying if $\gamma\mathcal{E} \subset H(a, b)$, for some half-space $H(a, b)$. However, it is known (Bland et al., 1981, Section 4) that $\gamma\mathcal{E} \subset H(a, b)$ if, and only if, $\alpha(\mathcal{E}, H) \leq -\gamma$. By applying this result, we can conclude that

$$\gamma\mathcal{E} \subset W \iff \max_{i \leq i \leq m} \alpha_i \leq -\gamma \text{ and } \max_{\|w\|=1} \alpha_w \leq -\gamma$$

where $\alpha_i = \alpha(\mathcal{E}, H(a_i, 0)) = z^T a_i / \sqrt{a_i^T P a_i}$ and $\alpha_w = \alpha(\mathcal{E}, H(w, 1)) = (z^T w - 1)/\sqrt{w^T P w}$, which concludes the proof. ■

A.11 Proof of Theorem 24

First, by the assumption of independent labeling among subspaces we have

$$\begin{aligned} p_{x,y} &= \mathbb{P}(H(x) = y \,|\, [H] \in \mathcal{V}_f) \\ &= \mathbb{P}(h_k(x^k) = y^k, \forall k \,|\, [h_k] \in \mathcal{V}_k, \forall k) \\ &= \prod_k \mathbb{P}(h_k(x^k) = y^k \,|\, [h_k] \in \mathcal{V}_k) \\ &= \prod_k p^k_{x,y^k} \end{aligned}$$

Therefore, we obtain

$$\begin{aligned} \sum_{y \in \mathcal{Y}_F} p^2_{x,y} &= \sum_{y_1 = \pm} \cdots \sum_{y_K = \pm} \prod_k (p^k_{x,y^k})^2 \\ &= \prod_k \sum_{y_k = \pm} (p^k_{x,y^k})^2 \\ &= \prod_k \left((p^k_{x,+})^2 + (p^k_{x,-})^2 \right) \\ &= \prod_k (1 - 2 p^k_{x,+} p^k_{x,-}) \end{aligned}$$

■

A.12 Proof of Theorem 25

For $p = (p_1, \cdots, p_K)$, let's define $f(p) = 1 - \prod_k (1 - 2p_k(1 - p_k))$ and $g(p) = \sum_k 2p_k(1 - p_k)$. Additionally, set $x^*_f = \arg\max_x f(p(x))$ and $x^*_g = \arg\max_x g(p(x))$, where $p(x) = (p_{x^1,+}, \cdots, p_{x^K,+})$. We wish to show that $f(x^*_g) \geq \frac{1}{K} f(x^*_f)$, from which our the main result follows from Theorem 13 of Golovin and Krause (2011).

It suffices to show $\frac{1}{K}g(p) \leq f(p) \leq g(p)$, for $p \in [0,1]^K$, since then

$$f(x_g^*) \geq \frac{1}{K}g(x_g^*) \geq \frac{1}{K}g(x_f^*) \geq \frac{1}{K}f(x_f^*)$$

Now, set $s_k = 1 - 2p_k(1-p_k) \in [\frac{1}{2}, 1]$, from which we obtain $f(s) = 1 - \prod_k s_k$ and $g(s) = K - \sum_k s_k$. By the arithmetic-geometric inequality we have

$$\prod_k s_k \leq \left(\prod_k s_k\right)^{1/K} \leq \frac{1}{K}\sum_k s_k,$$

which gives $f(s) \geq g(s)/K$.

For the second inequality, we proceed by induction on K. $K = 1$ is trivial. Now, assume that the inequality holds for K. It is enough to show that the function

$$h(t) = t\prod_{k=1}^{K} s_k - t - \sum_{k=1}^{K} s_k + K$$

is non-negative for $0 \leq t \leq 1$. Since $h'(t) = \prod_{k=1}^{K} s_k - 1 \leq 0$, $h(t)$ is a decreasing function. However, $h(1) = \prod_{k=1}^{K} s_k - 1 - \sum_{k=1}^{K} s_k + K \geq 0$ by the induction hypothesis, which concludes the proof. ■

Appendix B. Hit-and-Run Implementation

Let $W \subset \mathbb{R}^n$ be a convex body. The Hit-and-Run algorithm is a randomized algorithm for sampling a point $x \in W$ uniformly at random. More precisely, it generates a Markov Chain inside W which converges to the uniform distribution; starting at any given point $w_0 \in W$, it iteratively performs two steps:

1. Sample a direction vector D uniformly at random over the unit sphere
2. Set w_{t+1} as a random point on the line segment $\{w_t + sD,\ s \in \mathbb{R}\} \cap W$

Implementing step 1 can be easily done through the Marsaglia Algorithm (Marsaglia, 2007): simply sample $D \sim \mathcal{N}(0, I_d)$ and set $D \leftarrow D/\|D\|$.

As for step 2, the main difficulty is to find the intersections of the line $L = \{w_t + sD,\ s \in \mathbb{R}\}$ with the boundary of W. Although there are methods for finding these extremes for a general convex body W, we will focus on the particular case of $W = P \cap B_n$, where P is the polytope $P = \{w \in \mathbb{R}^n : Aw \leq 0\}$ and B_n is unit ball. In this case, it is easy to see that for the polytope:

$$\begin{aligned}
w_t + sD \in P &\iff A(w_t + sD) \leq 0 \\
&\iff sAD \leq -Aw_t \\
&\iff sa_i^T D \leq -a_i^T w_t, \text{ for each row } a_i \text{ of } A \\
&\iff \max_{a_i^T D < 0} -\frac{a_i^T w_t}{a_i^T D} \leq s \leq \min_{a_i^T D > 0} -\frac{a_i^T w_t}{a_i^T D}
\end{aligned}$$

Algorithm 5 Hit-and-Run algorithm

```
Input: convex body $W = \{w \in \mathbb{R}^n : \|w\| \leq 1 \text{ and } a_i^T w \leq 0,\ 1 \leq i \leq m\}$, any $w_0 \in W$,
    chain length $N$, inverse rounding transformation $T^{-1}$
Output: The Hit-and-Run chain $\{w_1, w_2, \dots, w_N\}$
1: $samples \leftarrow \{\}$
2: for $t$ from 0 to $N-1$ do
3:     $D \leftarrow T^{-1}\mathcal{N}(0, I_n)$
4:     $D \leftarrow D/\|D\|$
5:     $L_{pol}, U_{pol} \leftarrow \max_{a_i^T D < 0} -\frac{a_i^T w_t}{a_i^T D}, \min_{a_i^T D > 0} -\frac{a_i^T w_t}{a_i^T D}$
6:     $\Delta \leftarrow (w_t^T D)^2 - \|w_t\|^2 + 1$
7:     $L_{ball}, U_{ball} \leftarrow -w_t^T D - \sqrt{\Delta}, -w_t^T D + \sqrt{\Delta}$
8:     $L, U \leftarrow \max(L_{pol}, L_{ball}), \min(U_{pol}, U_{ball})$
9:     $s \leftarrow Unif([L, U])$
10:    $w_{t+1} \leftarrow w_t + sD$
11:    $samples \leftarrow samples \cup \{w_{t+1}\}$
12: end for
13: return $samples$
```

and, for the unit ball (keeping in mind that $\|D\| = 1$):

$$\begin{aligned} w_t + sD \in B_n &\iff \|w_t + sD\|^2 \leq 1 \\ &\iff s^2 + 2(w_t^T D)s + \|w_t\|^2 - 1 \leq 0 \\ &\iff -w_t^T D - \sqrt{\Delta} \leq s \leq -w_t^T D + \sqrt{\Delta} \end{aligned}$$

where $\Delta = (w_t^T D)^2 - \|w_t\|^2 + 1$. Note that $\Delta > 0$ since $w_t \in B_n$. Finally, we simply need to sample S uniformly from the intersection of the above two intervals and set $w_{t+1} = w_t + SD$.

One last point to consider is how the rounding transformation T affects the Hit-and-Run chain generation. Let $w_0 \in W$ be the chain's starting point, and let's define $w_0' = T(w_0) \in T(W)$. The usual Hit-and-Run algorithm over $T(W)$ gives rise to a chain $\{w_t'\}$, which is incrementally defined by $w_{t+1}' = w_t' + s_{t+1} D_{t+1}$. By applying T^{-1} on the previous equation, and setting $w_t = T^{-1}(w_t')$, we obtain a revised version of the Hit-and-Run update rule:

$$w_{t+1} = w_t + s_{t+1} T^{-1} D_{t+1}$$

In other words, the only necessary change is to multiply the random direction D by T^{-1}. Refer to Algorithm 5 for a pseudo-code of this entire process.

Appendix C. Rounding Implementation

In this section, we give more details on how to implement the optimized rounding algorithm described in Algorithm 3. In particular, we describe how to implement the *minimum_volume_ellipsoid* routine, and also how the rounding transformation T can be computed once we have the final rounding ellipsoid.

First, we consider the *minimum_volume_ellipsoid*$(\mathcal{E}, H)$ routine, as described in Goldfarb and Todd (1982). This function computes the minimum volume ellipsoid containing

Algorithm 6 Minimum Volume Ellipsoid (Goldfarb and Todd, 1982, Section 4)

```
Input: ellipsoid $\mathcal{E}(z, P = LDL^T)$ in $\mathbb{R}^n$, cutting half-space $H(a, b)$
Output: The minimum volume ellipsoid containing $\mathcal{E} \cap H$
 1: $\hat{a} \leftarrow L^T a$
 2: $\eta \leftarrow \sqrt{\sum_i d_i \hat{a}_i^2}$
 3: $\alpha \leftarrow (a^T z - b)/\eta$
 4: $p \leftarrow \frac{1}{\eta} D\hat{a}$
 5: $\tau \leftarrow (1 - n\alpha)/(n+1)$
 6: $z' \leftarrow z + \tau L p$
 7: $\delta \leftarrow n^2(1-\alpha^2)/(n^2-1)$
 8: $\sigma \leftarrow 2\tau/(1-\alpha)$
 9: $t_{n+1} \leftarrow \frac{n-1}{n+1}\frac{1-\alpha}{1+\alpha}$
10: for $j = n, n-1, \ldots, 1$ do
11:     $t_j \leftarrow t_{j+1} + \sigma p_j^2/d_j$
12:     $d'_j \leftarrow \delta d_j t_{j+1}/t_j$
13:     $\xi_j \leftarrow -\sigma p_j/(d_j t_{j+1})$
14:     $v_j^{(j)} \leftarrow p_j$
15:     for $r = j+1, \ldots, n$ do
16:         $L'_{rj} \leftarrow L_{rj} + \beta_j v_r^{(j+1)}$
17:         $v_r^{(j)} \leftarrow v_r^{(j+1)} + p_j L_{rj}$
18:     end for
19: end for
20: return $\mathcal{E}(z', P' = L'D'L'^T)$
```

$\mathcal{E} \cap H$, where $\mathcal{E}(z, P) = \{w \in \mathbb{R}^n : (w-z)^T P^{-1}(w-z) \leq 1\}$ is an ellipsoid and $H(a, b) = \{w \in \mathbb{R}^n : a^T w \leq b\}$ a cutting half-space. By defining $\alpha(\mathcal{E}, H) = (a^T z - b)/\sqrt{z^T P z}$, this computation can be separated into 3 special cases:

1. $\alpha(\mathcal{E}, H) \leq -\frac{1}{n}$: the minimum volume ellipsoid is $\mathcal{E}$ itself.
2. $\alpha(\mathcal{E}, H) \geq 1$: $\mathcal{E} \cap H$ is empty, and the minimum volume ellipsoid does not exist.
3. $-\frac{1}{n} < \alpha(\mathcal{E}, H) < 1$: the minimum volume ellipsoid exists and is different from $\mathcal{E}$. It can be computed via Algorithm 6.

In the particular case of our rounding algorithm, only the third case above has to be considered. In fact, the first case never happens since at every iteration we pick H satisfying $\alpha(\mathcal{E}, H) > -\gamma > -1/n$; as for the second case, it cannot happen because H is chosen to satisfy $W \subset \mathcal{E} \cap H$, which guarantees $\mathcal{E} \cap H \neq \emptyset$. Thus, as far as the rounding algorithm is concerned, we can directly apply Algorithm 6.

Another point to note is that Algorithm 6 only requires the LDL^T decomposition of the scaling matrices P as input, and it returns the same decomposition for the output ellipsoid. Although this algorithm could be implemented by using P directly, it can lead to numerical instabilities when computing $\eta = \sqrt{a^T P a}$; thus, it is preferred to use the LDL^T

decomposition formulation, which allows for a numerically stable implementation (Goldfarb and Todd, 1982).

One final point to consider is how to compute the inverse of the rounding transformation T^{-1}. From our discussion in Section 4.3, T is chosen as any linear transformation mapping the rounding ellipsoid $\mathcal{E}(z, P)$ into a ball of unit radius. Now, let's assume that P can be written as $P = JJ^T$, for some invertible matrix J. Then, we have

$$w \in \mathcal{E}(z, P) \iff (w - z)^T P^{-1}(w - z) \leq 1 \iff \|J^{-1}(w - z)\| \leq 1$$

The above equation implies that J^{-1} maps $\mathcal{E}$ into a unit ball, and we can choose $T^{-1} = J$. For example, we could pick $T^{-1} = LD^{1/2}$, where L and D are the factors in the LDL^T decomposition of P.

Appendix D. User Queries

Below, we can find all the user queries used for running experiments. We also provide the query selectivity (% of positive class) and dimensionality (# of distinct attributes).

The 11 SDSS queries used in our evaluation are:

Q1 (**2D**, 0.1%): $rowc \in (662.5, 702.5)$ AND $colc \in (991.5, 1053.5)$

Q2 (**2D**, 0.1%): $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 29^2$

Q3 (**2D**, 0.1%): $ra \in (190, 200)$ AND $dec \in (53, 57)$

Q4 (**2D**, 0.1%): $rowv^2 + colv^2 > 0.5^2$

Q5 (**4D**, 0.01%): $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 90^2$ AND $ra \in (180, 210)$ AND $dec \in (50, 60)$

Q6 (**6D**, 0.01%): $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 280^2$ AND $ra \in (150, 240)$ AND $dec \in (40, 70)$ AND $rowv^2 + colv^2 > 0.2^2$

Q7 (**4D**, 7.8%) : $x_1 > (1.35 + 0.25 * x_2)$ AND $x_3 + 2.5 * \log(2 * 3.1415 * petror50\_r^2) < 23.3$

Q8 (**7D**, 5.5%): $(dered\_r - dered\_i) < 2$ AND $cmodelmag\_i - extinction\_i \in [17.5, 19.9]$ AND $(dered\_r - dered\_i) - (dered\_g - dered\_r)/8. > 0.55$ AND $fiber2mag\_i < 21.7$ AND $devrad\_i < 20.$ AND $dered\_i < 19.86 + 1.60 * ((dered\_r - dered\_i) - (dered\_g - dered\_r)/8. - 0.80)$

Q9 (**5D**, 1.5%): $u - g < 0.4$ AND $g - r < 0.7$ AND $r - i > 0.4$ AND $i - z > 0.4$

Q10 (**5D**, 0.5%): $(g <= 22)$ AND $(u - g \in [-0.27, 0.71])$ AND $(g - r \in [-0.24, 0.35])$ AND $(r - i \in [-0.27, 0.57])$ AND $(i - z \in [-0.35, 0.70])$

Q11 (**5D**, 0.1%): $((u - g > 2.0)$ OR $(u > 22.3))$ AND $(i \in [0, 19])$ AND $(g - r > 1.0)$ AND $((r - i < 0.08 + 0.42 * (g - r - 0.96))$ OR $(g - r > 2.26))$ AND $(i - z < 0.25)$

Additionally, the 18 Car queries are the following:

Q1 (**6D**, 0.32%): $class$ = 'minivan' AND $price\_msrp \leq 30000$ AND $length > 5$ AND $length * width > 10.1$ AND $fuel\_type$ = 'regular unleaded' AND $transmission \neq$ '8-speed shiftable automatic' AND $transmission \neq$ '9-speed shiftable automatic'

Q2 (**8D**, 0.27%): $price\_msrp \leq 22132$ AND $basic\_year \geq 5$ AND $drivetrain\_year \geq 10$ AND $horsepower > 156$ AND $body\_type \neq$ 'suv' AND $transmission$ = '6-speed shiftable automatic' AND $year \geq 2016$ AND $fuel\_tank\_capacity \geq 65$

Q3 (**8D**, 0.27%): $year \geq 2016$ AND $length * height * width \geq 15.0$ AND $basic\_year \geq 4$ AND $class$ = 'full-size car' AND $price\_msrp < 100000$ AND $engine\_type$ = 'gas'

Q4 (**6D**, 0.25%): *body_type* = 'truck' AND *height* $\geq$ 1.9 AND *torque* $\geq$ 3800 AND *price_msrp* $\leq$ 30000 AND *year* = 2017 AND *base_engine_size* $\geq$ 5
Q5 (**6D**, 0.23%): *class* = 'full-size van' AND *body_type* = 'van' AND *engine_type* = 'gas' AND *model* = 'nv cargo' AND *price_msrp* < 27000 AND *horsepower* < 300
Q6 (**4D**, 0.34%): *height* < 1.51 AND *drivetrain_year* $\geq$ 10 AND *transmission* $\neq$ '6-speed manual' AND *transmission* $\neq$ '7-speed manual' AND *class* = 'subcompact car'
Q7 (**6D**, 0.82%) : *class* = 'mid-size car' AND *transmission* = '6-speed shiftable automatic' AND *drivetrain_year* > 5 AND *price_msrp* < 29000 AND *basic_km* $\geq$ 80467 AND *body_type* $\neq$ 'suv'
Q8 (**9D**, 0.32%): *length* $\geq$ 6 AND *body_type* $\neq$ 'sedan' AND *fuel_type* $\neq$ 'premium unleaded (recommended)' AND *drive_type* $\neq$ 'front wheel drive' AND *fuel_type* $\neq$ 'premium unleaded (required)' AND *basic_year* $\geq$ 4 AND *drivetrain_year* $\geq$ 5 AND *price_msrp* < 32000 AND *height* > 2.5
Q9 (**4D**, 0.30%): (*body_type* = 'truck' OR *body_type* = 'van') AND *price_msrp* < 30000 AND *height* $\geq$ 2.5 AND *length* > 6
Q10 (**8D**, 0.36%): *body_type* = 'truck' AND *horsepower* > 350 AND *drive_type* = 'four wheel drive' AND *engine_type* $\neq$ 'diesel' AND *fuel_tank_capacity* > 100 AND *year* $\geq$ 2017 AND *suspension* $\neq$ 'stabilizer bar stabilizer bar' AND *price_msrp* < 35000
Q11 (**6D**, 0.29%): (*body_type* = 'suv' OR *body_type* = 'truck') AND (*drive_type* = 'all wheel drive' OR *drive_type* = 'four wheel drive') AND *height* > 1.8 AND *price_msrp* < 33000 AND *length* > 5.9 AND *suspension* LIKE '%independent%'
Q12 (**4D**, 0.20%): *price_msrp* < 25000 AND *horsepower* > 200 AND *year* = 2017 AND *length* > 5.5
Q13 (**10D**, 0.29%): *price_msrp* < 23000 AND *basic_year* $\geq$ 5 AND *drivetrain_year* $\geq$ 10 AND *basic_km* $\geq$ 80000 AND *drivetrain_km* $\geq$ 100000 AND *engine_type* IN ('gas', 'hybrid') AND *body_type* IN ('sedan', 'hatchback') AND *drive_type* = 'front wheel drive' AND *fuel_tank_capacity* $\geq$ 55 AND *year* $\geq$ 2017
Q14 (**5D**, 0.23%): *price_msrp* < 13000 AND *drive_type* = 'front wheel drive' AND *transmission* LIKE '%manual%' AND *length* < 4.5 AND *horsepower* < 110
Q15 (**7D**, 0.91%): *transmission* LIKE '%automatic%' AND *price_msrp* < 25000 AND *class* NOT LIKE '%pickup%' AND *class* NOT LIKE '%suv%' AND *basic_year* $\geq$ 4 AND *year* $\geq$ 2017 AND *height* $\leq$ 1.62
Q16 (**5D**, 0.21%): *price_msrp* < 26000 AND *body_type* IN ('van', 'truck') AND *height* < 2.5 AND *height* > 2 AND *basic_year* > 3
Q17 (**6D**, 0.20%): *horsepower* > 150 AND *year* = 2017 AND *make* IN ('hyundai', 'honda') AND *length* < 4.5 AND *engine_type* IN ('gas', 'diesel') AND *price_msrp* < 20000
Q18 (**6D**, 0.20%): *price_msrp* < 30000 AND *body_type* IN ('sedan', 'suv') AND *engine_type* = 'hybrid' AND *year* = 2017 AND *basic_year* $\geq$ 5 AND *drivetrain_year* $\geq$ 10

References

Stephen H. Bach, Bryan He, Alexander Ratner, and Christopher Ré. Learning the structure of generative models without labeled data. In *International Conference on Machine Learning*, volume 70, pages 273–282, 2017.

Claude J. P. Bélisle, H. Edwin Romeijn, and Robert L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18(2): 255–266, 1993. ISSN 0364-765X.

Robert G. Bland, Donald Goldfarb, and Michael J. Todd. Feature article—the ellipsoid method: A survey. *Operations Research*, 29(6):1039–1091, 1981. doi: 10.1287/opre.29.6.1039.

Lawrence Bull, Keith Worden, Graeme Manson, and Nikolaos Dervilis. Active learning for semi-supervised structural health monitoring. *Journal of Sound and Vibration*, 437:373 – 388, 2018. ISSN 0022-460X. doi: 10.1016/j.jsv.2018.08.040.

Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *International Conference on Machine Learning*, volume 28, pages 160–168, 2013.

Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17*, pages 337–344, 2005.

Daniele De Martino, Matteo Mori, and Valerio Parisi. Uniform sampling of steady states in metabolic networks: Heterogeneous scales and rounding. *PLoS ONE*, 10(4):e0122670, 2015. ISSN 19326203. doi: 10.1371/journal.pone.0122670.

Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 517–528, 2014.

Kyriaki Dimitriadou, Olga Papaemanouil, and Yanlei Diao. AIDE: An active learning-based approach for interactive data exploration. *TKDE*, 28(11):2842–2856, 2016.

Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Query-by-committee made real. In *Advances in Neural Information Processing Systems 18*, pages 443–450, 2006.

Donald Goldfarb and Michael J. Todd. Modifications and implementation of the ellipsoid algorithm for linear programming. *Mathematical Programming*, 23(1):1–19, 1982. doi: 10.1007/BF01583776.

Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011. doi: 10.1613/jair.3278.

Alon Gonen, Sivan Sabato, and Shai Shalev-Shwartz. Efficient active learning of halfspaces: an aggressive approach. *Journal of Machine Learning Research*, 14(1):2583–2615, 2013.

Simone Hantke, Zixing Zhang, and Björn W. Schuller. Towards intelligent crowdsourcing for audio data annotation: Integrating active learning in the real world. In *INTERSPEECH*, pages 3951–3955, 2017.

Robbert L. Harms and Alard Roebroeck. Robust and fast Markov chain Monte Carlo sampling of diffusion MRI microstructure models. *Frontiers in Neuroinformatics*, 12:97, 2018. ISSN 1662-5196. doi: 10.3389/fninf.2018.00097.

Enhui Huang, Liping Peng, Luciano Di Palma, Ahmed Abdelkafi, Anna Liu, and Yanlei Diao. Optimization for active learning-based interactive database exploration. *Proceedings of the VLDB Endowment*, 12(1):71–84, 2018.

Brendan Juba and Hai Le. Precision-recall versus accuracy and the role of large data sets. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4039–4048, 2019.

Haya Kaspi, Sean P. Meyn, and Richard L. Tweedie. Markov chains and stochastic stability. *Journal of the American Statistical Association*, 92(438):792, 1997. ISSN 01621459. doi: 10.2307/2965732.

Akshay Krishnamurthy, Alekh Agarwal, Tzu-Kuo Huang, Hal Daumé, and John Langford. Active learning for cost-sensitive classification. In *International Conference on Machine Learning*, volume 70, pages 1915–1924, 2017.

William A. Link and Mitchell J. Eaton. On thinning of chains in MCMC. *Methods in Ecology and Evolution*, 3(1):112–115, 2012. ISSN 2041210X. doi: 10.1111/j.2041-210X.2011.00131.x.

Wenzhao Liu, Yanlei Diao, and Anna Liu. An analysis of query-agnostic sampling for interactive data exploration. *Communications in Statistics – Theory and Methods*, 47 (16):3820–3837, 2017.

Lászlo Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*, volume 50 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1986. ISBN 9781611970203.

Lászlo Lovász. Hit-and-run mixes fast. *Mathematical Programming*, 86(3):443–461, 1999. ISSN 0025-5610. doi: 10.1007/s101070050099.

George Marsaglia. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, 2007. ISSN 0003-4851. doi: 10.1214/aoms/1177692644.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X.

Daniela Pohl, Abdelhamid Bouchachia, and Hermann Hellwagner. Batch-based active learning: Application to social media data for crisis management. *Expert Systems with Applications*, 93:232 – 244, 2018. ISSN 0957-4174. doi: 10.1016/j.eswa.2017.10.026.

Alexander J. Ratner, Christopher M. De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems 29*, pages 3567–3575, 2016.

Burr Settles. *Active Learning*. Morgan & Claypool Publishers, 2012. ISBN 1608457257.

Hyunjune S. Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, page 287–294, 1992. doi: 10.1145/130385.130417.

Jinhua Song, Hao Wang, Yang Gao, and Bo An. Active learning with confidence-based answers for crowdsourcing labeling tasks. *Knowledge-Based Systems*, 159:244 – 258, 2018. ISSN 0950-7051. doi: 10.1016/j.knosys.2018.07.010.

Simon Tong and Daphne Koller. Support Vector Machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

Kirill Trapeznikov, Venkatesh Saligrama, and David Castañón. Active boosted learning (ActBoost). In *International Conference on Artificial Intelligence and Statistics*, volume 14, pages 743–751, 2011.

Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *Proceedings of the VLDB Endowment*, 12(3):223–236, 2018.

Lin Yang, Yizhe Zhang, Jianxu Chen, Siyuan Zhang, and Danny Chen. Suggestive annotation: A deep active learning framework for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 399–407, 2017.