A Factorized Version Space Algorithm for "Human-in-the-Loop" Data Exploration

Luciano Di Palma†, Yanlei Diao*,†, Anna Liu*

†École Polytechnique, France
*University of Massachusetts Amherst, USA

November 11, 2019
Motivation

- Data is growing extremely fast
- Human ability to comprehend data remains limited

**Annual Size of the Global Datasphere**

Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework
Active Learning-based Interactive Data Exploration (IDE)
– in an “explore-by-example” framework

Database
\[ x_1, x_2, x_3, \ldots \]

Active Learner

User

inspect the unlabeled data

\[ < x_1, ? > \]

\[ < x_1, y_1 > \]
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework

![Diagram illustrating the process of active learning-based data exploration. The database contains unlabeled data, and the user interacts with the active learner to label data, guiding the exploration process.]
Active Learning-based Interactive Data Exploration (IDE) – in an "explore-by-example" framework
Active Learning-based Interactive Data Exploration (IDE) – in an “explore-by-example” framework

Active learning-based framework requires fewer labeled examples than a traditional learner
Focus of our work

Slow convergence problem

Existing AL algorithms and state-of-the-art IDE systems may require hundreds of labeled examples to reach high accuracy.

Figure: Comparison of explore-by-example IDE methods (SDSS 6D, 0.01%)
Idea #1: Explore Version Space algorithms

- **Main feature**: strong theoretical guarantees on performance.\(^1\)
- **Main drawbacks**:
  - Usually too expensive for the IDE scenario.
  - In practice, still suffers from slow convergence in high-dimensional settings.

**Question**
Can we leverage their strong theoretical results, while still running under interactive performance?

**Our solution**
We propose an **optimized** version space algorithm meeting the interactive performance requirements.

More details in our paper!

The bisection rule

- $\mathcal{H}$: set of classifiers $h : \mathcal{X} \rightarrow \mathcal{Y}$ (e.g. linear, svm, ...)
- **Version Space**: subset of $h \in \mathcal{H}$ consistent with the labeled data $\mathcal{L}$:

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

- **Main idea**: shrink $\mathcal{V}$ as quickly as possible.

**Bisection Rule**

Let $\mathcal{V}_{x,y} = \{ h \in \mathcal{V} : h(x) = y \}$. The bisection rule selects $x^*$ satisfying:

\[
x^* \in \arg \max_x 1 - \sum_{y \in \mathcal{Y}} p_{x,y}^2
\]

where $p_{x,y} = \frac{\text{vol}(\mathcal{V}_{x,y})}{\text{vol}(\mathcal{V})}$.

---

\textsuperscript{2}Gonen and al., *Efficient Active Learning of Halfspaces: an Aggressive Approach*, JMLR, 2013
Idea #2: add factorization information

The user decision-making process can usually be broken into a set of simple yes or no questions.

**Example:** consider a user who wishes to buy a car. The user may have the following questions in mind:

1. Is gas mileage good enough?
2. Is the vehicle spacious enough?
3. Is the color a preferred one?

**Question**

Can we leverage this decomposition information to help us expedite our algorithm’s convergence?
Factorization: formalism

- $Q$: user’s unknown interest query, involving attributes $\{A_1, \ldots, A_d\}$
- Based on intuition, $Q$ is broken down into sub-queries $Q_1, \ldots, Q_K$, with each $Q_k$ involving a subset of attributes $A^k = \{A_{k1}, \ldots, A_{kd_k}\}$.
- $(A^1, \ldots, A^K)$ is called the factorization structure.
- The final query $Q$ is related to $Q_k$ by a boolean function $F$:

$$Q(x) = F(Q_1(x^1), \ldots, Q_K(x^K))$$

- For each $x$, the user must provide the partial labels $Q_1(x^1), \ldots, Q_K(x^K)$ in a consistent and independent fashion.

This decomposition lets us model each sub-query $Q_k$ directly, breaking down the original classification task into several, simpler classification tasks.
Given a factorization structure \((A^1, \ldots, A^K)\), we define:

- **Factorized hypothesis space**: we model one classifier per subspace:
  \[
  \tilde{\mathcal{H}} = \mathcal{H}^1 \times \ldots \times \mathcal{H}^K
  \]
  where each tuple \(H = (h_1, \ldots, h_K)\) is treated as a multi-label classifier:
  \[
  x \rightarrow (h_1(x^1), \ldots, h_K(x^K)) \in \{-, +\}^K
  \]

- **Factorized version space**: by applying the usual definition, we can show the version space to be:
  \[
  \tilde{\mathcal{V}} = \mathcal{V}^1 \times \ldots \times \mathcal{V}^K
  \]
  where \(\mathcal{V}^k = \{h \in \mathcal{H}^k : h(x^k) = y^k, \text{ for } (x, y) \in \mathcal{L}\}\).
A Factorized Bisection Rule

Applying the bisection rule over \( \tilde{\mathcal{V}} \) translates to:

\[
x^* \in \arg \max_x 1 - \sum_{y \in \{-, +\}^K} p_{x, y}^2
\]

where \( p_{x, y} = \frac{\text{vol}(\tilde{\mathcal{V}}_{x, y})}{\text{vol}(\tilde{\mathcal{V}})} \).

**Theorem**

The above computation can be simplified to:

\[
x^* \in \arg \max_x 1 - \prod_{k} \left( 1 - 2p_{x^k, +}(1 - p_{x^k, +}) \right)
\]

where \( p_{x^k, +} = \frac{\text{vol}(\mathcal{V}_{x^k, +}^k)}{\text{vol}(\mathcal{V}^k)} \).

In other words, we simply have to repeat the usual version space computations for each subspace.
Theoretical results

- $A_f$: any Active Learner making use of the partial labels information.
- $H = (h_1, \ldots, h_K)$: classifiers matching the user preference in each subspace.
- $\text{cost}(A_f, H)$: # of queries that $A_f$ takes to identify $h_k$ in every subspace.
- $\text{cost}(A_f)$: average cost of $A_f$ across all possible labeling $H$.

Theoretical guarantees on performance

The Factorized Version Space algorithm satisfies:

$$\text{cost}(\text{FactVS}) \leq OPT_f \cdot \left(1 + \sum_k \ln \left( \frac{1}{\min h_k \pi_k(h_k)} \right) \right)^2$$

where $OPT_f = \min_{A_f} \text{cost}(A_f)$.
Experimental Evaluation

- **Datasets**
  - Sloan Digital Sky Survey (SDSS)
    - a large sky survey database
    - 1% sample: 1.9 million tuples, 4.9GB
    - user interest queries from the SDSS query release

- **Algorithms for comparison**
  - Factorized VS
  - DSM (w/ factorization)
  - AL algorithms: ALuMa, Simple Margin, Query-by-Disagreement

- **Total number of queries**: 11

---

5 Support Vector Machine Active Learning with Applications to Text Classification, JMLR, 2001
Experimental results: classification accuracy

- **Q6 (6D, 0.01%)**: $(a_1 - 682.5)^2 + (a_2 - 1022.5)^2 < 280^2 \text{ AND } b_1 \in (150, 240) \text{ AND } b_2 \in (40, 70) \text{ AND } c_1^2 + c_2^2 > 0.2^2$

- **Q10 (5D, 0.5%)**: $g \leq 22 \text{ AND } u - g \in (-0.27, 0.71) \text{ AND } g - r \in (-0.24, 0.35) \text{ AND } r - i \in (-0.27, 0.57) \text{ AND } i - z \in (-0.35, 0.70)$

![Graph showing F-score vs Iteration for Fact VS, DSM, SM, ALuMA, and QBD algorithms.](a) Q6 (6D, 0.01%) (b) Q10 (5D, 0.5%)

The Factorized VS algorithm outperforms both DSM and AL algorithms for high-dimensional exploration.
Experimental results: running time

Figure: Time per iteration for Q10
Conclusion and Future Work

Our main take-aways are:

- We successfully extended version space algorithms to the IDE scenario, including several optimizations and extension to factorization.
- Our algorithms significantly outperform the existing state-of-the-art AL algorithms and IDE systems in accuracy and convergence speed, while maintaining the per-iteration time within a couple seconds.

Future work:

- Address **inconsistent labeling** by extending our models to a probabilistic one.
- Relax the assumption that a factorization structure is provided before the exploration beginning.
- Learn the factorization structure on-the-fly.
Thank you! Questions?

**Code:** https://gitlab.inria.fr/ldipalma/aideme
Bisection rule: theoretical guarantees

- Let $\mathcal{A}$ be an AL algorithm, and $h$ a classifier matching the user interest. We define $\text{cost}(\mathcal{A}, h)$ as the number of queries that $\mathcal{A}$ takes to identify $h$. Additionally, we set:

$$
\text{cost}(\mathcal{A}) = \sum_{h} \text{cost}(\mathcal{A}, h) \pi(h)
$$

- **Theorem**: the VS bisection rule satisfies:

$$
\text{cost(VS bisection)} \leq \text{OPT} \cdot \left( 1 + \ln \frac{1}{\min_h \pi(h)} \right)^2
$$

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

---

Every point $x$ splits $\mathcal{V}$ into two sets:

$$\mathcal{V}_{x,\pm} = \{ h \in \mathcal{V} : h(x) = \pm \}$$

The bisection rule looks for $x^*$ satisfying:

$$x^* \in \arg\min_x \left| \frac{vol(\mathcal{V}_{x,+})}{vol(\mathcal{V})} - 0.5 \right|$$

Since computing volumes is hard, we fall back to sampling:

$$p(x) = \frac{vol(\mathcal{V}_{x,+})}{vol(\mathcal{V})} \approx \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}(H_i(x) = +)$$

where $\{H_i\}_{i=1}^{M}$ is a random sample from $\mathcal{V}$. 

Sampling the version space: linear case

- Let’s consider the case of homogeneous linear classifiers:
  \[ \mathcal{H}_{hom} = \{ h_w(x) = \text{sign}(w^T x), \text{ where } \|w\| \leq 1 \} \]

- Given a labeled set \( \mathcal{L} = \{(x_i, y_i)\} \), the version space is:
  \[ \mathcal{V}_{hom} = \{ h_w \in \mathcal{H}_{hom} : y_i x_i^T w \geq 0, \text{ for all } (x_i, y_i) \in \mathcal{L} \} \]

- Sampling from \( \mathcal{V}_{hom} \) amounts to sample \( w \) from:
  \[ W = \{ w \in \mathbb{R}^d : \|w\| \leq 1 \land Aw \geq 0 \} \]
  where \( A_i = y_i x_i \) is the \( i \)-th row of \( A \).
Sampling from $\mathcal{W}$: the Hit-and-Run algorithm

- **Hit-and-Run**: algorithm for sampling from a convex body
- It generates a Markov Chain converging to the **uniform distribution** over the convex body

*Figure*: Hit-and-Run algorithm illustration
Optimizing Hit-and-Run: rounding

**Problem:** mixing time can be very large when convex set is elongated.

**Figure:** Rounding illustration
Effect of version space optimizations

- **Q2 (2D, 0.1%)**: \((rowc - 682.5)^2 + (colc - 1022.5)^2 < 29^2\)
- **Q3 (2D, 0.1%)**: \(ra \in (190, 200)\) AND \(dec \in (53, 57)\)

**Figure**: Effect of sampling optimizations on f-score and time

The proposed optimizations improve accuracy and reduce computation time.
State-of-the-art comparison: no factorization

- **Q2 (2D, 0.1%)**: $(\text{rowc} - 682.5)^2 + (\text{colc} - 1022.5)^2 < 29^2$
- **Q3 (2D, 0.1%)**: $r_a \in (190, 200)$ AND $d_{ec} \in (53, 57)$

---

**Figure**: Comparison of AL methods for 2 selected queries

Our optimized VS algorithm is effective against the cold-start problem.