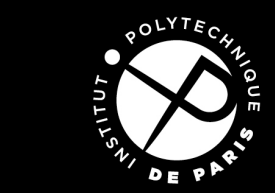


INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX043

Thèse de doctorat

New Algorithms and Optimizations for Human-in-the-Loop Model Development

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École Doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 07 / 07 / 2021, par

LUCIANO DI PALMA

Composition du Jury :

Sihem AMER-YAHIA Directrice de recherche, CNRS (LIG)	Présidente / Rapporteuse
Michele SEBAG Directrice de recherche, CNRS (LRI)	Rapporteuse
Madalina FITERAU Assistant professor, UMass Amherst (Information Fusion Lab)	Examinatrice
Ping MA Distinguished professor, University of Georgia (Dept of Statistics)	Examinateur
Yanlei DIAO Professor, Ecole Polytechnique (LIX)	Directrice de thèse
Anna LIU Professor, UMass Amherst (Dept of Mathematics and Statistics)	Co-directrice de thèse
Peter HAAS Professor, UMass Amherst (DREAM)	Invité
Themis PALPANAS Directeur de Recherche, Université de Paris (DIIP)	Invité

Résumé

Avec l'avènement de l'ère du big data, l'écart ne cesse de se creuser entre la quantité d'informations disponibles et la capacité humaine à en tirer un contenu de grande valeur. Cette situation crée un besoin d'outils capables de combler le fossé entre l'homme et les données et de guider efficacement les utilisateurs à travers ces grands bassins de données pour les aider à en tirer des informations précieuses.

Pour mieux illustrer le besoin actuel de tels outils de navigation de données et de découverte d'informations, nous considérons deux scénarios pratiques : l'*exploration de bases de données* et l'*annotation de données*. L'exploration de bases de données est le cas d'un utilisateur qui souhaite explorer une grande base de données et récupérer des objets d'intérêt. Cependant, les systèmes de bases de données traditionnels ne prennent souvent pas en charge ces tâches d'*exploration de données*, ce qui oblige les utilisateurs à recourir à un processus d'exploration manuelle qui prend beaucoup de temps. En d'autres termes, les utilisateurs de bases de données ne disposent pas des moyens nécessaires pour explorer efficacement leurs propres ensembles de données et en extraire le contenu pertinent. Une autre situation difficile découle d'une récente tendance industrielle connue sous le nom de «Machine Learning for Everyone». Les entreprises informatiques proposent des plateformes en nuage pour aider les utilisateurs à développer des modèles d'apprentissage automatique sur leurs ensembles de données et à en tirer des informations utiles. Cependant, ces modèles dépendent généralement de volumes importants de *données annotées* pour l'apprentissage, ce qui nécessite souvent un effort manuel considérable de la part des annotateurs humains et, éventuellement, une connaissance spécifique du domaine. Par conséquent, le développement de modèles est souvent peu pratique pour les utilisateurs réguliers car ils n'ont pas les moyens d'annoter automatiquement leurs grands ensembles de données de manière efficace et précise.

Pour surmonter ces défis, nous proposons dans cette thèse un cadre de travail *«human-in-the-loop» pour le développement efficace de modèles sur de grands ensembles de données*. Dans ce cadre, nous visons à construire un modèle de classification de l'intérêt de l'utilisateur en nous appuyant sur le feedback de l'utilisateur sur des exemples de données ciblées ; dans ce contexte, l'intérêt de l'utilisateur pourrait se référer aux objets de données recherchés dans l'exploration de la base de données ou aux étiquettes inconnues du scénario d'annotation des données. À cette fin, nous appliquons des algorithmes d'Active Learning pour sélectionner une petite séquence d'instances de données que l'utilisateur doit étiqueter et

dériver un modèle précis tout en offrant des performances interactives en présentant l'instance de données suivante.

Cependant, les techniques d'Active Learning existantes ne parviennent souvent pas à fournir des performances satisfaisantes lorsqu'elles sont construites sur de grands ensembles de données. Non seulement ces modèles nécessitent souvent des centaines d'instances de données étiquetées pour atteindre une précision élevée (*convergence lente*), mais la recherche de la prochaine instance à étiqueter peut prendre beaucoup de temps (*inefficacité*), ce qui est incompatible avec la nature interactive du processus d'exploration humain. Pour résoudre les problèmes de convergence lente et d'inefficacité, cette thèse repose sur deux idées principales. Premièrement, nous introduisons un nouvel algorithme d'Active Learning basé sur le *Version Space* pour les classificateurs à noyau, qui présente de fortes garanties théoriques sur la convergence et permet également une implementation efficace en temps et en espace. Deuxièmement, en tirant parti des connaissances supplémentaires obtenues dans le processus d'exploration et d'étiquetage de l'utilisateur, nous explorons une nouvelle possibilité de *factoriser un algorithme d'Active Learning* pour effectuer l'exploration dans un ensemble de sous-espaces de faible dimension, ce qui accélère encore la convergence et réduit l'effort d'étiquetage de l'utilisateur.

Plus précisément, cette thèse comprend les contributions suivantes :

1. *Un nouveau cadre théorique pour les algorithmes de Version Space (VS) sur les classificateurs à noyau.* Nous proposons un nouveau cadre théorique qui permet une implémentation efficace de la stratégie Generalized Binary Search sur les classificateurs à noyau. Notre cadre offre à la fois de solides garanties théoriques sur les performances et une mise en œuvre efficace en temps et en espace par rapport aux travaux précédents. Nous prouvons également des limites d'erreur de généralisation sur la précision et le F-score, ce qui permet à nos techniques de fonctionner sur un échantillon du grand ensemble de données original avec une perte de performance minimale.

2. *Un algorithme de Version Space optimisé.* Sur la base de nos résultats théoriques, nous proposons un algorithme VS optimisé appelé OptVS qui utilise l'algorithme hit-and-run pour échantillonner l'espace des versions. Cependant, le hit-and-run peut nécessiter des milliers d'itérations pour produire un échantillon de haute qualité, ce qui peut entraîner un coût élevé en temps de calcul. Pour réduire ce coût, nous développons une série d'optimisations de l'échantillonnage pour améliorer à la fois la qualité de l'échantillon et le temps d'exécution. En particulier, nous fournissons une version très efficace de la technique de *rounding* pour améliorer la qualité de l'échantillon du Version Space.

   En pratique, nous observons qu'OptVS atteint des performances similaires ou supérieures à celles des algorithmes de Version Space sur 26 des 29

modèles d'utilisateurs considérés, tout en tournant en dessous d'une seconde par itération. Par exemple, pour un cas particulier d'utilisateur, notre algorithme atteint un score F de 67% après 25 points étiquetés, alors que les autres algorithmes sont toujours à 19% ou moins.

3. *Un algorithme de Version Space factorisé.* En outre, nous proposons un nouvel algorithme qui exploite la structure de factorisation fournie par l'utilisateur pour créer des sous-espaces et factoriser l'Version Space pour effectuer de l'Active Learning dans les sous-espaces. Par rapport aux travaux récents qui ont également utilisé la factorisation pour l'Active Learning, notre travail l'explore dans le nouveau cadre des algorithmes VS et élimine les fortes hypothèses faites, telles que la convexité des modèles d'intérêt de l'utilisateur, ce qui entraîne une amélioration significative des performances tout en augmentant l'applicabilité dans les problèmes du monde réel. Nous fournissons également des résultats théoriques sur l'optimalité de notre algorithme VS factorisé et des optimisations pour traiter les variables catégorielles.

   Nos résultats d'évaluation montrent que, pour les 25 patterns d'utilisateurs considérés, notre algorithme VS factorisé surpasse les Active Learners non factorisés ainsi que DSM, un autre algorithme prenant en compte la factorisation, souvent par une large marge tout en maintenant la vitesse interactive. Par exemple, pour un modèle d'utilisateur particulier testé, notre algorithme atteint un F-score de plus de 80% après 100 itérations d'étiquetage alors que DSM est encore à 40% et les autres algorithmes VS à 10% ou moins.

4. *Un algorithme d'apprentissage pour les classificateurs factorisés.* En suivant le raisonnement intuitif derrière le processus de décision de l'utilisateur, nous développons un nouvel algorithme de classification appelé le Factorized Linear Model (FLM), qui décompose son processus prédictif en une collection de tâches de classification indépendantes. Plus précisément, le FLM modélise l'intérêt de l'utilisateur comme une combinaison d'objets convexes de faible dimension, ce qui permet d'obtenir un classificateur précis, efficace et interprétable. En pratique, nous observons que le classificateur FLM atteint des performances comparables ou supérieures à celles du SVM et d'un autre modèle interprétable, le VIPR, pour 80% de tous les modèles d'intérêt de l'utilisateur, tout en prenant seulement quelques minutes pour s'entraîner sur un grand ensemble de données de près d'un million de points.

5. *Le Swapping Algorithm pour une exploration efficace des données.* En appliquant notre nouveau classificateur factorisé au scénario d'Active Learning, nous développons une nouvelle stratégie d'Active Learning factorisé automatique appelée le Swapping Algorithm. Cette technique utilise initialement OptVS pour échapper à la convergence lente des itérations initiales,

puis passe à une stratégie basée sur FLM pour profiter de sa précision de classification supérieure. Notre évaluation montre que l'algorithme de permutation atteint des performances similaires ou supérieures à celles des active learners non factorisés pour tous les patterns d'utilisateurs considérés, tout en se rapprochant des méthodes explicitement factorisées. Par exemple, dans le cas d'un modèle d'utilisateur particulier, l'algorithme de permutation et notre stratégie VS factorisée atteignent une précision presque parfaite après 150 itérations d'étiquetage, alors que les autres techniques sont encore à 70% de F-score et moins.

Abstract

With the advent of the big data era, there is an ever-increasing gap between the amount of available information and the human ability to derive high-value content from it. This situation creates a need for tools that can bridge the human-data gap and efficiently guide users through these large pools of data to help them draw valuable insights.

To better exemplify the current need for such data navigation and insight discovery tools, we consider two practical scenarios: *database exploration* and *data annotation*. Database exploration considers the case of a user who wishes to explore a large database and retrieve objects of interest. However, traditional database systems often do not support such *data exploration* tasks, forcing users to resort to a time-consuming manual exploration process. In other words, database users lack the means to efficiently explore their own data sets and extract relevant content from them. Another challenging situation arises from a recent industry trend known as "Machine Learning for Everyone". IT companies deliver cloud platforms to help users develop machine learning models over their data sets and draw useful data-driven insights. However, such models usually depend on high volumes of *annotated data* for training, which often requires considerable manual effort from human annotators and, possibly, specific domain knowledge. Consequently, model development is often impractical for regular users since they lack the means to automatically annotate their large data sets in an efficient and accurate fashion.

To overcome these challenges, in this thesis, we propose a *human-in-the-loop framework for efficient model development over large data sets*. In this framework, we aim to build a classification model of the user interest by relying on the user feedback over targeted data examples; in this context, user interest could refer to the data objects sought by the user in database exploration or the unknown labels of the data annotation scenario. To this end, we apply active learning algorithms to select a small sequence of data instances for the user to label and 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.

However, existing active learning techniques often fail to provide satisfactory performance when built over large data sets. Not only do such models often require hundreds of labeled data instances to reach high accuracy (*slow convergence*), but retrieving the next instance to label can be time-consuming (*inefficiency*), making it incompatible with the interactive nature of the human exploration process. To address the slow convergence and inefficiency issues, this

thesis embodies two main ideas. First, we introduce a novel *version-space*-based active learning algorithm for kernel classifiers, which has strong theoretical guarantees on convergence also allows for an efficient implementation in time and space. Second, by leveraging additional insights obtained in the user exploration and labeling process, we explore a new opportunity to *factorize an active learner* to perform exploration in a set of low-dimensional subspaces, which further expedites convergence and reduces the user labeling effort.

More specifically, this thesis includes the following contributions:

1. *A New Theoretical Framework for Version Space (VS) Algorithms over Kernel Classifiers.* We propose a new theoretical framework that allows for an efficient implementation of the Generalized Binary Search strategy over kernel classifiers. Our framework offers both strong theoretical guarantees on performance and efficient implementation in time and space compared to previous work. 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. *An Optimized Version Space Algorithm.* Based on our theoretical results, we propose an optimized VS algorithm called OptVS that uses the hit-and-run algorithm 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 for improving the sample quality from the version space.

   In practice, we observe that OptVS achieves similar or better performance than state-of-the-art version space algorithms for 26 out of the 29 user patterns considered while running under 1 second per iteration at all times. For example, for a particular user pattern considered, our algorithm achieves a 67% F-score after 25 labeled points while other algorithms are still at 19% or below.

3. *A Factorized Version Space Algorithm.* 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 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 and optimizations for dealing with categorical variables.

Our evaluation results show that, for all 25 user patterns considered, our factorized VS algorithm outperforms non-factorized active learners as well as DSM, another factorization-aware algorithm, often by a wide margin while maintaining interactive speed. For example, for a particular user pattern tested, our algorithm achieves an F-score of over 80% after 100 labeling iterations while DSM is still at 40% and other VS algorithms at 10% or lower.

4. *A Learning Algorithm for Factorized Classifiers.* Following the intuitive reasoning behind the user decision-making process, we develop a new human-inspired classification algorithm, called the Factorized Linear Model (FLM), that decomposes its predictive process as a collection of independent classification tasks. More precisely, the FLM models the user interest as a combination of low-dimensional convex objects, resulting in an accurate, efficient, and interpretable classifier. In practice, we observe that the FLM classifier achieves comparable or better performance than SVM and another interpretable model, VIPR, over 80% of all user interest patterns while taking only a few minutes to train over a large data set of nearly one million points.

5. *A Swapping Algorithm for Efficient Data Exploration.* By applying our new factorized classifier to the active learning scenario, we develop a novel automatically factorized active learning strategy called the Swapping Algorithm. This technique initially employs OptVS to escape the slow convergence of initial iterations and then swaps to an FLM-based strategy to take advantage of its higher classification accuracy. Our evaluation shows that the Swapping Algorithm achieves similar or better performance than non-factorized active learners for all considered user patterns while approximating the explicitly factorized methods. For example, in the case of a particular user pattern, both the Swapping Algorithm and our factorized VS strategy achieve nearly perfect accuracy after 150 labeling iterations, while other techniques are still at 70% F-score and below.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, data is being generated at an unprecedented rate. A recent survey [Reinsel et al., 2018] estimates that humans create more than 2.5 quintillion bytes of data every day in the forms of photos, videos, and social media posts, and these numbers are rapidly increasing as time passes by; so much that 90% of the total world data was created in the past two years. In contrast, the human ability to understand data remains almost unchanged, creating an ever-increasing disparity between the amount of available information and our ability to derive high-value content from it. Therefore, enabling users to efficiently navigate through such large pools of data and help them draw valuable insights is becoming an increasingly crucial, albeit challenging, task.

To exemplify these new challenges brought by the big data era, we next investigate two scenarios where the human-data disparity is becoming an issue:

Database Exploration. Real-world applications often employ a database system for efficient storage and management of large data collections. Databases are designed to provide a simple interface between human users and application data, enabling efficient access, retrieval, and modifications of stored content. However, despite being heavily optimized for data retrieval, these systems cannot efficiently support data exploration tasks. In this scenario, the user may only possess an intuitive understanding of his/her own interest and very limited knowledge of the data distribution, making it impossible to accurately translate his/her interest set into a database query. Consequently, in such cases, the user must resort to *manual data exploration*, an extremely tedious and time-consuming process: besides evaluating several queries, the user also needs to closely inspect each result set to decide whether further refinement is necessary. See Figure 1.1 for an example of this process. In conclusion, traditional database systems are unable to help users efficiently explore their large collections of data, creating *a need for automated data exploration tools capable of efficiently guiding users towards their interest set.*t

Data Annotation. In a recent industry trend known as "Machine Learning

```
SELECT * FROM Cars WHERE  price < 30000
SELECT * FROM Cars WHERE  20000 < price < 30000  AND body_type = 'sedan'
SELECT * FROM Cars WHERE  20000 < price < 28000  AND body_type = 'sedan'
SELECT * FROM Cars WHERE  20000 < price < 25000  AND body_type = 'sedan'
SELECT * FROM Cars WHERE  20000 < price < 25000  AND body_type = 'sedan'  AND year = 2018
SELECT * FROM Cars WHERE  20000 < price < 28000  AND body_type = 'sedan'  AND year = 2018
```

Figure 1.1: Example from our user studies [Huang et al., 2018, 2019] of how the human user explores a Car database for objects of interest.

for Everyone", IT companies deliver cloud platforms such as SageMaker[1] and Google AutoML[2] to help users train machine learning models over their data sets with minimum effort. These platforms help users draw insights from their data by treating it as a *model learning* problem; for example, by leveraging the purchase data of clients over an online store, a machine learning model can learn to group users accordingly to their interests and provide better product recommendations. However, a key concern with machine learning models is their need for large volumes of high-quality training data: although data itself is largely available for most applications, annotated data is often much scarcer. Current industry solutions for this "lack of labeled data" problem are often limited to crowdsourcing or manual labeling by dedicated IT teams, but both approaches are far from ideal. On one hand, crowdsourcing is ineffective when domain expertise is needed, which is the case for healthcare or scientific data; on the other hand, building a dedicated labeling team does not scale to the millions of data sets and users that these cloud platforms support. As such, to make model learning more practical for users, there is *an increasing need for tools that can accurately annotate large sets of data in an automated fashion.*

In summary, database users are missing tools for automated data exploration, while machine learning platforms lack the efficient means for annotating large data sets. To address both issues, in this thesis we propose a *human-in-the-loop framework for efficient model development over large data sets.* In model development, one aims to build an accurate classification model of the user interest by relying on the user feedback over targeted data examples. To gain an intuitive understanding of this framework, let us revisit the data exploration and annotation issues. In data exploration, by asking the user to label a selection of database objects as "interesting" or not, we can build a classifier of the user interest and use it to retrieve all relevant records from the database. Similarly, in the data annotation scenario, by asking the user to annotate a well-chosen subset of data points, we can train an accurate classifier capable of automatically labeling all remaining elements in the large data set.

In what follows, we discuss the technical challenges brought by the interactive model development approach, including a detailed description of our proposed

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

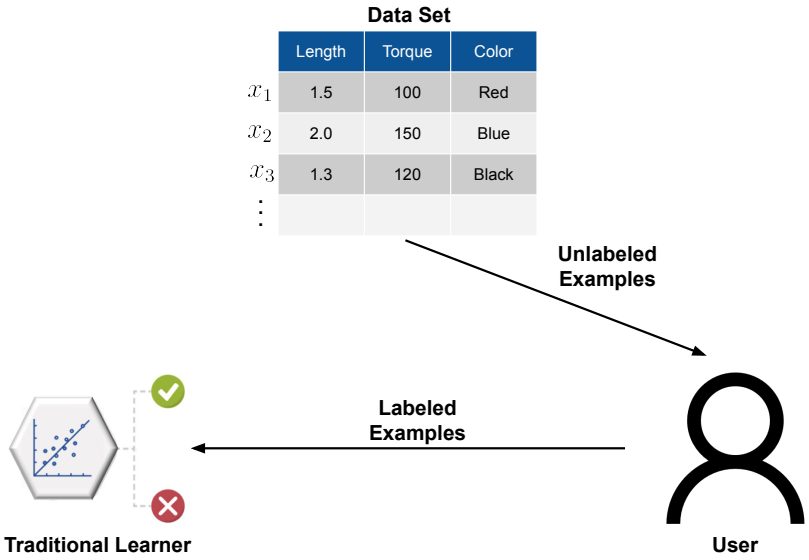


Figure 1.2: Illustration of the interactive model development framework.

contributions to mitigate these problems.

1.1 Technical Challenges of Interactive Model Development

To make *interactive model learning* practical for large data sets, there are two main problems to overcome:

- *Minimizing user effort*: Our framework requires the user to annotate a subset of data points to build an accurate classifier. However, in practice, the user has limited patience and may not be willing to review too many examples. Therefore, to guarantee the usefulness of our framework, *our techniques and algorithms must primarily focus on minimizing the user labeling effort*; in other words, we wish to train a high-accuracy classifier with as few labeled examples as possible.
- *Performance requirements*: The user should spend as little time idle as possible; in other words, classification models should be quick to train, and users should wait as little time as possible for the data points to label.

To address the first issue, we consider a related subset of the Machine Learning domain known as *Active Learning* [Tong and Koller, 2001, Trapeznikov et al., 2011, Settles, 2012, Gonen et al., 2013]. In this area, the objective is to build an accurate classification model over a data set with minimal labeled data. More

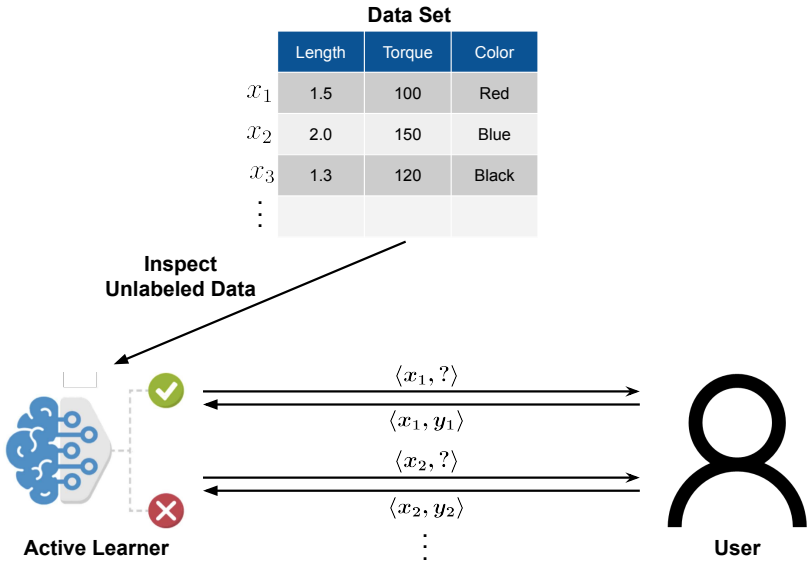


Figure 1.3: Illustration of active-learning-based model development.

precisely, an active learning strategy is capable of inspecting the data source and selecting a data point that, once labeled, will provide the fastest increase in classification accuracy. In other words, active learning is a human-in-the-loop framework for model learning where the human annotator is continuously inquired for the labels of targeted data points, resulting in an accurate classifier built with minimum effort. In particular, these algorithms are especially suited for the model development scenario and will become the basis for our selection strategies.

However, existing active learning techniques 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] and a recent active-learning-based data exploration system [Huang et al., 2018] require the user to label over 200 instances to learn a classification model over 6 attributes with an F-score of 80%.[3] Asking the user to label hundreds of elements to achieve high accuracy, known as the slow convergence problem, is undesirable.

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

[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.

techniques embodies two main ideas: *Version Space algorithms* (Section 1.2) and *Factorization* (Section 1.3). Additionally, in Section 1.4, we also discuss a new human-inspired classification model based on the human decision-making process, bringing several advantages to modeling human-centered learning tasks.

1.2 Version Space Algorithms

In our efforts to develop an active learning strategy matching the needs of the interactive model development scenario, we aim to leverage a particular class of techniques with strong theoretical properties called *Version Space algorithms*. The version space $\mathcal{V}$ [Mitchell, 1982] is defined as the set of all classifiers $h \in \mathcal{H}$ consistent with the user-labeled data, that is, $h \in \mathcal{V} \iff h(x) = y$ for every user labeled point (x, y). As the user labels more and more data points, classifiers are constantly eliminated from the version space $\mathcal{V}$ until a single classifier perfectly matching the user preference remains. Version space algorithms aim to explore this property by devising query strategies that reduce $\mathcal{V}$ as quickly as possible, thus minimizing the number of data points labeled by the user.

One particularly relevant version space strategy is the Generalized Binary Search (GBS) algorithm [Dasgupta, 2005, Golovin and Krause, 2011], also known as the *bisection rule*. As the name suggests, this technique manages to approximately halve the version space at every iteration, which guarantees an exponentially fast reduction of the set of compatible classifiers. More interestingly, the bisection rule was shown to possess near-optimal theoretical guarantees on performance. More precisely, let us define $cost(\mathcal{A})$ as the average number of queries an active learner $\mathcal{A}$ will take to identify the classifier matching the user's interest. Then, one can show that $cost(GBS)$ is bounded by $\min_{\mathcal{A}} cost(\mathcal{A})$, the minimum cost among all active learners, times a log factor depending on the complexity of the set of classifiers $\mathcal{H}$. Due to these theoretical properties, the GBS strategy offers a good starting point for designing active learning techniques capable of combating the slow start problem.

Despite its theoretical advantages, the GBS algorithm is prohibitively expensive to run in practice due to the exponential size of the version space [Sauer, 1972]. In the literature, several approximations were proposed to counter this limitation, each offering a different trade-off in convergence speed, time per iteration, and theoretical guarantees. However, existing methods are still not able to match the needs of the interactive model development scenario. As an example, let's consider a few state-of-the-art version space algorithms: Simple Margin [Tong and Koller, 2001], Query-by-Disagreement (QBD) [Settles, 2012], Kernel Query-by-Committee (KQBC) [Gilad-Bachrach et al., 2006], and ALuMA [Gonen et al., 2013]. On one hand, Simple Margin and QBD are rough approximations of the GBS strategy and are empirically observed to suffer from slower convergence speeds. On the other hand, KQBC and ALuMA can better approximate the

bisection rule via a sampling procedure, but these methods can be very time-consuming. For example, ALuMA runs at every iteration a hit-and-run sampling procedure with thousands of steps to sample a single hypothesis, which is then repeated to generate hundreds of hypotheses. On top of that, among all presented techniques, ALuMA is the only one with provable theoretical guarantees on performance.

With the above considerations in mind, in this thesis, we aim to develop a novel version space strategy that matches the fast convergence and interactive performance needs of the model development scenario. More precisely, we propose the following major contributions:

- *A new theoretical framework for version space algorithms over kernel classifiers* (Chapter 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 framework offers both strong theoretical guarantees on performance and 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 method for efficient estimation of 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.

- *Implementation and Optimizations* (Chapter 4): Based on our theoretical results, we propose an optimized version space algorithm called OptVS. Similar 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 the 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.

- *Evaluation* (Section 4.4): Our evaluation results show that OptVS OptVS achieves similar or better performance than 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], for 26 out of the 29 user patterns considered. For a particular set of workloads, we observed that OptVS managed to reach a 67% F-score after 25 labeled points, while other algorithms were still at 19% or below. In terms of running time, OptVS takes less than one second per iteration at all times, being around 10 times faster than ALuMA.

1.3 Factorization Structure

To make version space 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 simpler questions whose answers can be combined to derive the final label. For example, when a customer decides whether a car model is of interest, he may have a set of 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?

While the user may have a high-level intuition about how each question is related to a subset of attributes, he may not be able to specify these questions precisely. In fact, doing so would require precise knowledge of the relationship between the attributes composing each question, as well as any constants and thresholds. For example, the user may know a vehicle's space is a function of the body type, length, width, and height, but he probably does not know how these values are precisely connected, nor he knows which volume threshold precisely captures his interest.

The above insight allows us to design new version space algorithms that leverage the high-level intuition a user has for decomposing the decision-making process, formally called a *factorization structure*, to combat the slow start problem. More precisely, we develop the following contribution:

1. *A Factorized Version Space Algorithm* (Chapter 5): 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.

2. *Factorized exploration strategies* (Sections 5.3–5.5): Our factorized algorithm also supports three different selection strategies: Greedy, Squared-Loss, and Product-Loss strategies. In particular, the first two losses come with strong theoretical guarantees on performance, while the last one offers a better convergence speed in practice.

3. *Optimization for categorical subspaces* (Section 5.6): Additionally, we have also developed a new optimization for categorical subspaces that replaces

the VS sampling scheme with efficient memorization of categories and their labels, leading to a 100% improvement in running time in some cases.

4. *Evaluation* (Section 5.7): Using two real-world data sets and a large suite of user interest patterns, our evaluation results show that, for every user pattern considered, our factorized VS algorithm outperforms non-factorized active learners like OptVS and Simple Margin [Tong and Koller, 2001], 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 an F-score of over 80% after 100 labeling iterations while DSM is still at 40% and other VS algorithms at 10% or lower.

1.4 Learning a Factorized Classification Model

Despite the performance advantages brought by a factorization structure, in practice, the user may not always be able to provide one at the beginning of the exploration process. Doing so would require a deep understanding of his own decision-making process (e.g. what makes a car interesting or not?) and familiarity with the data distribution (e.g. what is a reasonable price range for a sedan?), which may not be available before annotating several data examples. Because of this limitation, a natural question to consider is: *can an accurate factorization structure be learned from the user-labeled data*? In such cases, the users could be freed from the burden of providing a factorization and let it become an extra learning step to the system while only requiring a moderately higher number of examples to be labeled.

More generally, one can consider the question: *can we learn a classification model which mimics the user decision-making process under factorization*? Recent results from our user studies [Huang et al., 2018, 2019] suggest that when users are asked to retrieve data points of interest from large data sets, they often break the labeling process into several simpler questions whose answers, when combined, derive the final label. This observation suggests the existence of a human-inspired classification model capable of decomposing its predictive process into a collection of low-dimensional, independent classification tasks. In theory, such an *automatically factorized classifier* can offer several benefits:

- *Accuracy*: In the same way that explicitly factorized active learners provide a significant performance boost when compared to non-factorized strategies, we expect an automatically factorized classifier to bring similar performance benefits in human-centered learning tasks.
- *Efficiency*: Training can also be made efficient as the original complex learning problem is broken into a combination of simpler ones.

- *Interpretability*: By mimicking the decision-making of human users, the predictive process of a factorized classifier should be relatively easy to interpret by the human practitioner.

Based on the above considerations, in this thesis, we propose the following contributions:

- *A Learning Algorithm for Factorized Classifiers* (Section 6.1): Following the intuitive reasoning behind the human decision-making process, we develop a new human-inspired classification algorithm, called the Factorized Linear Model (FLM), that decomposes its predictive process as a collection of independent classification tasks. More precisely, the FLM models the user interest as a combination of low-dimensional polytopes, resulting in a classifier that is accurate, efficient, and interpretable by humans. Additionally, we propose a few extensions that enhance the applicability of our technique, including an extension to kernel classifiers and optimizations for categorical attributes.
- *A Swapping Algorithm for Efficient Data Exploration* (Section 6.2): By applying our new classification model to the active learning scenario, we have developed a new exploration strategy called the Swapping Algorithm. This technique initially employs OptVS to escape the slow convergence of initial iterations and then swaps to an FLM-based strategy to take advantage of its higher classification accuracy. We also develop a couple of techniques that enable a slow transition between the two strategies and avoid any drop in performance due to the change in the classification model.
- *Evaluation* (Sections 6.3 and 6.4): In the batch learning scenario, our evaluation shows that the FLM classifier achieves comparable or better performance than SVM and another interpretable model over 80% of the considered user interest patterns while managing to train over nearly one million data points in a couple of minutes. In the active learning case, we observed that the Swapping Algorithm outperforms OptVS, our baseline non-factorized active learner, while approximating the explicitly factorized Fact VS, a less realistic upper bound which relies on extra user information. In the particular case of a complex SDSS pattern, both the Swapping Algorithm and Fact VS achieved nearly perfect accuracy after 150 labeling iterations, while other techniques were still at 70% F-score and below.

1.5 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 contains a literature review on the most relevant topics for our work, including version space

algorithms and data exploration for model development. Chapter 3 develops a new version space algorithm over kernel classifiers with strong theoretical guarantees on performance, while Chapter 4 offers an efficient implementation of the said algorithm through several sampling optimizations. Next, Chapter 5 extends our version space techniques to support a factorization structure from the user, including three factorized querying strategies and optimization for categorical subspaces. Additionally, Chapter 6 develops a new human-inspired classification algorithm that mimics the user decision-making process under factorization, including a learning algorithm for the factorization structure and applications to both batch and active learning cases. Finally, Chapter 7 contains a summary of all thesis contributions, as well as a discussion on possible lines of future work.

Chapter 2

Literature Review

In this chapter, we survey the most relevant works for each of our proposed contributions, including version space algorithms, interactive data exploration systems, and interpretable machine learning.

2.1 Active Learning and Version Space Algorithms

In this first section, we provide an overview of the active learning domain and, in particular, version space techniques.

2.1.1 Active Learning

Settles [2012] and Monarch [2021] provide an overview of the most relevant results in the active learning (AL) literature. We focus our attention on a particular form called pool-based active learning, in which a classification model is built over a large data set by relying on user feedback over targeted data samples. More precisely, based on the current classification model, the active learning algorithm can inspect the data pool and extract the data example that, if labeled, will result in the fastest increase in classification accuracy. In doing so, active learners can build an accurate classifier with few labeled examples and, consequently, minimal user effort.

In practice, AL is usually employed in domains where labeling data is expensive and labeled data sets are scarce. For example, recent fields 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 intend to apply active learning, in particular the version space algorithms, in the setting of human-in-the-loop model development.

2.1.2 The Version Space

Mitchell [1977, 1982] was the first to introduce the concept of Version Space (VS). In his work, Mitchell considered the *concept learning* problem, in which one aims to identify the true concept (hypothesis) $h^* \in \mathcal{H} \subset \{h : x \mapsto y \in \{\pm 1\}\}$ from its evaluations over a finite set of examples $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m$ with $y_i = h^*(x_i)$. To determine h^*, Mitchell introduced a special subset of classifiers called the version space $\mathcal{V}$, defined as the set of all hypotheses perfectly matching the training data:

$$\mathcal{V} = \{h \in \mathcal{H} : h(x_i) = y_i \text{ for all } (x_i, y_i) \in \mathcal{L}\} \tag{2.1}$$

The main interest of considering the version space is it possesses two main properties:

- Assuming no labeling mistakes in $\mathcal{L}$, the true concept h^* must be inside $\mathcal{V}$, that is, $h^* \in \mathcal{V}$.
- The version space $\mathcal{V}$ tends to shrink as more labeled data becomes available.

From these two properties, one can interpret the version space $\mathcal{V}$ as the set of all *candidate hypotheses* for h^*. In particular, if enough labeled data are available, the version space is reduced to a single element that must correspond to the true concept, that is, $\mathcal{V} = \{h^*\}$. Additionally, in case the available labeled data is not enough to identify h^*, Mitchell [1982, 1997] also proposed two relevant ideas:

- *Majority Voting*: We can approximate h^* by a majority voting among candidate hypotheses $h \in \mathcal{V}$.
- *Labeling new examples*: Another option is to add new labeled examples to $\mathcal{L}$. In this case, Mitchell proposes to label the objects x over which the *candidate hypotheses disagree the most*. This way, each new labeled example would eliminate a large portion of candidates from $\mathcal{V}$ and, consequently, help quickly identify h^*.

In recent years, the idea of version space for tackling concept learning tasks and, in general, supervised learning problems has largely been discarded because of its natural limitations. First, version-space-based algorithms like Find-S and Candidate Elimination [Mitchell, 1982, 1997] need to maintain some representation of $\mathcal{V}$, which can be exponential in size even for simple concepts [Haussler, 1988, Sauer, 1972]. In addition, its intolerance to label noise makes it impractical to most use cases; even though a few works had some success in proposing a noise-tolerant VS strategy [Hirsh, 1990, Hong and Tseng, 1997], they can still suffer from the exponential size problem. However, despite its limitations, VS-based techniques have still found relative success in the Active Learning scenario, which we discuss next.

2.1.3 Version Space Methods in Active Learning

One domain where version-space-based techniques have found their success is in Active Learning (AL). Based on the groundwork set by Mitchell [1982, 1997], several AL strategies have been devised to explore the version space properties. More precisely, these techniques aim to reduce the version space $\mathcal{V}$ as quickly as possible until we are left with a single hypothesis achieving perfect classification. Various strategies have been developed to estimate the VS reduction of each sample, each offering a different trade-off in terms of performance, efficiency, and theoretical guarantees. Below, we give an overview of the most relevant algorithms in the AL literature:

Generalized Binary Search (GBS) [Dasgupta, 2005, Golovin and Krause, 2011]: Also called the *Version Space Bisection* rule, this algorithm implements the data labeling strategy proposed by Mitchell [1977, 1982], searching for a point x for which the classifiers in $\mathcal{V}$ disagree the most. More precisely, let us define $\mathcal{V}^{x,y} = \{h \in \mathcal{V} : h(x) = y\}$ and let π be any probability distribution over the set of classifiers corresponding to our prior knowledge about the user interest (e.g. uniform). Then, the GBS strategy selects the point x for which the sets $\mathcal{V}^{x,y}$ have approximately the same size for all labels, that is, it minimizes $\max_y \pi(\mathcal{V}^{x,y})$.

The main interest behind GBS comes from its strong theoretical guarantees. If we define the cost of any active learner $\mathcal{A}$ as the average number of queries required to reduce $\mathcal{V}$ to a single element (and, consequently, identifying the user interest), then one can show that

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

where $OPT = \min_{\mathcal{A}} cost(\mathcal{A})$. Because of this, GBS is said to possess *near-optimal performance guarantees*.

However, implementing the bisection rule is prohibitively expensive: for each instance x in the data set, one must evaluate $h(x)$ and $\pi(h)$ for each hypothesis h in the version space, which is often exponential in size $\mathcal{O}(m^D)$, where m is the size of the data set and D is the VC-dimension [Sauer, 1972]. Due to this limitation, several approximations of the bisection rule have been introduced in the literature.

Simple Margin [Tong and Koller, 2001]: Simple Margin is a rough approximation of the bisection rule for SVM classifiers, leveraging the heuristic that data points close to the SVM's decision boundary closely bisect the version space $\mathcal{V}$. However, this method can suffer from slow convergence in practice, especially when $\mathcal{V}$ is very asymmetric. In addition, Simple Margin does not possess any theoretical guarantees on performance.

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 samples two random hypotheses from the version space, while QBD trains one positive and one negatively biased classifier. Again, these methods do not provide any theoretical results on performance and, on top of that, they can suffer from slow convergence since the selected point is only guaranteed to "cut" $\mathcal{V}$, but possibly not by a significant 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. Additionally, ALuMA was also show to possess similar theoretical guarantees on performance as the original GBS algorithm. Finally, ALuMA was 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]. In this work, we focus on how version space techniques can be applied to the interactive model development problem.

2.2 Data Exploration and Model Development

Next, we consider other forms of interactive model development in the literature, including a recent line of work on human-in-the-loop data exploration systems.

Data Programming under Weak Supervision. One form of interactive 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 can build 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] was designed to automatically generate labeling functions from an initial labeled data set, leading to improved classification accuracy and further reduction of the user manual effort. However, this system still requires an initial labeled set in the order of hundreds to thousands of labeled instances, which is nontrivial to obtain, especially in the new trend of "Machine Learning for Everyone" where an everyday user may start

with no labeled data at all.

Multi-Criteria Decision Aid. Another interesting form of human-in-the-loop data exploration comes from the Multi-Criteria Decision Aid (MCDA) framework [Figueira et al., 2005, Nemery and Ishizaka, 2013]. In MCDA, the main objective is to design a system capable of helping users make *complex decisions* over large data sets when facing *multiple conflicting criteria*. More precisely, the user has to specify a collection of functions $\{u_i\}_i$ computing relevance scores for each data object x, and the system will help the user decide on which elements are the best according to the specified criteria. For example, consider the case of a user looking to buy a car. In this case, the user could specify a few criteria, such as the vehicle's price, speed, and size, and then ask the system to find the cars offering the best trade-off according to these metrics.

One popular approach to MCDA is to rely on the Pareto optimality criteria [Grierson, 2008, Talukder and Deb, 2020]. More precisely, these methods focus on retrieving the data points x on the Pareto front defined by $\{u_i(x)\}_i$ since they cannot be dominated by any other object in the data set. However, such methods still require the user to further investigate and refine the set of Pareto optimal elements, which may be non-trivial to do for a high number of criteria [Talukder and Deb, 2020]. Another common approach to MCDA involves Multi-Attribute Utility Theory (MAUT) [Nemery and Ishizaka, 2013, Bresson et al., 2020] methods. Its main idea is to train, with the user's help, an aggregation function A that combines all criteria into a single value condensing the overall utility of a data object. However, one problem with these methods lies in interpretability: finding the right balance between the interpretability of the utility score and the representation power of A is a non-trivial issue. One approach to this problem is to consider the Choquet integral aggregator [Bresson et al., 2020], which can model interactions between criteria while maintaining some degree of interpretation by the user.

Interactive Data Exploration. In the interactive data exploration domain, the main objective is to design a system that guides the user towards discovering relevant data points in a large data set. One recent line of work is "explore-by-example" systems [Dimitriadou et al., 2014, 2016, Liu et al., 2017, Huang et al., 2018], which leverage active learning techniques to construct an accurate model of the user interest from a small set of user-labeled data points. These human-in-the-loop systems require minimizing both the user labeling effort and the running time to find a new data point for labeling in each iteration. 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 is assumed to form a convex object in the data space. By leveraging this property, this work proposes a dual-space model (DSM) that builds a polytope model of the user interest on top of any active learning strategy. In particular,

this polytope model partitions the data space into three segments: the *positive region*, where points are guaranteed to be interesting, the *negative region*, where points are guaranteed to be uninteresting, and the remaining *unknown region*. In doing so, DSM can use the polytope model to automatically label most of the data points selected by the active learner and, consequently, significantly reduce the user labeling effort.

In addition, DSM can also leverage a *factorization structure* provided by the user in a limited form. When the user pattern is a conjunction of convex or concave sub-patterns, their system factorizes the data space into low-dimensional subspaces and runs the dual-space model in each subspace. In doing so, the original complex exploration process is broken into several simpler tasks, thus significantly improving the performance of their system.

2.3 Interpretable Models and Feature Selection

This section explores the Machine Learning and Statistics literature for works that could help us develop a classification model mimicking the user decision-making process under factorization, as described in Section 1.4. In particular, we focus our attention on two main lines of work: Interpretable Machine Learning and Feature Selection.

Interpretable Machine Learning. In the Interpretable Machine Learning domain, the main objective is to design machine learning models whose predictive processes can be easily interpreted by humans. More precisely, besides striving for accurate predictions, interpretable models aim to provide people a better understanding of "how" and "why" predictions are made. In particular, the factorized classifier we aim to build is also interpretable given that it mimics the reasoning of real users, making this line of work very suitable for drawing inspiration from. In what follows, we focus our attention on a few most relevant works in this domain, leaving a complete survey on the topic to Molnar [2019].

- *Logistic Regression*: Logistic Regression is one of the simplest and most studied classification models in the literature. It starts by assuming that the positive and negative classes can be well-separated by a linear hyperplane $b+\langle x, w\rangle$, and constructs a probabilistic model of the label $Y \in \{\pm 1\}$ on the form $\mathbb{P}(Y = y|X = x) = \sigma\left(y(b + \langle x, w\rangle)\right)$ with σ being the sigmoid function. Due to its simplicity, the logistic regression classifier can be relatively easy to interpret: not only it leads to linear decision boundaries, but its weights also represent a rough measure of the feature importance. However, this model's simplicity can also be a disadvantage since it becomes unable to accurately learn more complex interest patterns: in fact, none of the user patterns we observed in practice are linearly separable; see Appendix D for more information.

- *Decision Rules* [Holte, 1993, Letham et al., 2015]: Decision rules attempt to break the user interest pattern as a collection of simple IF-THEN conditions. For example, over a dataset containing cars, this algorithm may generate conditions on the form "IF price $> 100k$ THEN $Y = 0$" or "IF color = red AND horsepower > 100 THEN $Y = 1$". As we can see, decision rules are simple to interpret by humans since their decisions correspond to combinations of simple logical formulas. Algorithms can also improve interpretability by reducing the number of features composing each rule [Holte, 1993] and minimizing conflicts between different rules [Letham et al., 2015]. However, rule-based learning also has some fundamental disadvantages: first, most models cannot deal with numerical attributes and require an initial discretization step; finally, IF-THEN rules usually fail to describe linear relationships between features and output and often require the combination of several rules to compensate for this deficiency.

- *Projection Retrieval for Classification (PRC)* [Fiterau and Dubrawski, 2012]: PRC methods assume that labels can be accurately inferred from a small selection of features and aim to find which sets of attributes are more relevant for different data points. In other words, PRC algorithms compute a mapping between data points and low-dimensional projections, effectively partitioning the data into sets that can be accurately classified with a small number of targeted features. Similar to our factorized classifier idea, PRC also attempts to break the complex user labeling process into a collection of simple, low-dimensional decisions. However, both methods still differ in their execution of this idea: PRC models the labeling process as a *single* low-dimensional decision for each point, while factorization models the label of any point as a *set* of several low-dimensional decisions.

In summary, we can achieve interpretability through very different means. Some methods bet on a simple relationship between features and output, while others attempt to break a complex decision as a collection of easy-to-interpret classifiers. However, despite having similar motivations, none of the above techniques directly addresses the factorization of the user decision-making process as observed in our user studies.

Feature Selection. Feature selection methods attempt to find a small set of most relevant features for prediction while having minimal impact on performance. In the factorization-learning scenario, although these methods cannot directly deal with multiple subspaces of features (at least not without the subspace labels), we expect that some form of feature selection will be necessary to induce a low-dimensional representation in each subspace. In particular, we will focus our attention on the following lines of work:

- *Lasso-based penalization* [Tibshirani, 1996]: In high-dimensional scenarios, the interpretability of linear classifiers becomes compromised by the high

number of parameters to evaluate. Lasso-based penalties were introduced to counter this problem: by including an L_1-based penalty term to the classification loss, the optimal weight vector is guaranteed to be *sparse*, thus effectively removing irrelevant features from consideration. In more recent times, Lasso-based penalization has been successfully applied in a wide range of scenarios [Hastie et al., 2015] such as Compressed Sensing and Matrix Completion.

- *Group Lasso* [Yuan and Lin, 2006]: In some applications, one may be interested in selecting whole groups of features instead of individual ones. In such cases, the Lasso-based penalization scheme has been extended into the Group Lasso, an L_2-based penalty function that allows for naturally grouping different sets of features. This grouped selection property is particularly important for selecting categorical variables [Chiquet et al., 2016] once they are encoded into groups of numerical features.

- *Importance-based selection* [Breiman, 2001, Strobl et al., 2007]: Another common feature selection technique is to rank features based on their *importance*. Importance is often calculated through some particular property of the model under consideration. For example, linear classifiers define importance as the absolute value of its weights, while decision trees measure the mean decrease in impurity for each feature. However, some methods can also compute importance in a model-agnostic fashion. *Importance permutation* [Strobl et al., 2007], for instance, measures the impact on the performance caused by randomly permuting the values of each feature; intuitively, more important features should lead to a bigger impact on performance. Overall, feature-importance methods can be reasonably effective in practice and also efficient to run.

Chapter 3

Generalized Binary Search over Kernel Classifiers

In this chapter, 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 is usually not known 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 the entire data set, 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 setting. Another drawback of this work is the lack of theoretical guarantees on convergence speed.

In what follows, we introduce a new version-space 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 satisfy the following condition:

Axiom 3.1 (Realizability or Separability). *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 $\hat{\mathcal{H}} \coloneqq \mathcal{H} \,/\, \sim$, representing our prior knowledge over which hypotheses are more likely to match the user's preferences. In the absence of prior knowledge, a uniform prior can be assumed.

The *version space* [Mitchell, 1982] 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 rule* searches for the point x such that all the sets $\mathcal{V}^{x,y}$ have approximately the same probability mass:

Definition 3.2 (Generalized Binary Search - GBS). *Let $\mathcal{V}$ denote the version space at any iteration of the active learning loop. For any $(x, y) \in \mathcal{X} \times \mathcal{Y}$, we define the **cut probabilities**:*

$$p_{x,y} = \mathbb{P}(h(x) = y \,|\, [h] \in \mathcal{V}) = \frac{\pi(\mathcal{V}^{x,y})}{\pi(\mathcal{V})} \tag{3.1}$$

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

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

One main advantage of the GBS strategy is it enjoys near-optimal performance guarantees [Golovin and Krause, 2011]. 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

$$cost(GBS) \leq OPT \cdot \left(1 + \log \frac{1}{\min_h \pi([h])}\right)^2 \tag{3.2}$$

with $OPT = \min_{\mathcal{A}} cost(\mathcal{A})$. In other words, when simply considering the number of iterations until convergence, the GBS strategy differs from the optimum by at most a log factor. In particular, in the case were π is uniform and $\mathcal{H}$ has finite VC-dimension D, Sauer's Lemma [Sauer, 1972] allows us to write $\log(1/\min_h \pi([h])) = \log|\hat{\mathcal{H}}| = \mathcal{O}(D \log N)$, which discloses a logarithmic dependency on the number of data points N.

3.2 Parameterized Hypothesis Space

In the following sections, we would like to work with classifiers parameterized by some vector $\theta \in \mathbb{R}^p$ (for instance, linear classifiers). In these cases, it is preferable to work directly with the parameters themselves instead of the classifiers, especially when devising algorithms.

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

Definition 3.3 (Parameterization). *A parameterization is any surjective function $\psi : \Omega \ni \theta \to h_\theta \in \mathcal{H}$ assigning a parameter $\theta \in \Omega \subset \mathbb{R}^p$ to every hypothesis $h \in \mathcal{H}$.*

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 3.4 (Parameterized Version Space). *Let $\psi : \Omega \to \mathcal{H}$ be a parameterization, and let $\mathcal{L}$ be a set of labeled 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 with a parameterized set of hypotheses is we can easily define a prior π over the hypothesis classes through a probability distribution over the parameter space. To see this, let $p_0(\theta)$ be any probability distribution over Ω representing a prior over parameters. Then, we can define π as:

$$\pi([h']) \coloneqq \mathbb{P}_{\theta \sim p_0}(h_\theta \sim h') \tag{3.3}$$

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

Theorem 3.5. *Let $\psi : \Omega \to \mathcal{H}$ be a parametrization, and let $p_0(\theta)$ be any prior over Ω. Denote by GBS_θ the greedy strategy selecting*

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

with $p_{x,y} = \mathbb{P}_{\theta \sim p_0}(h_\theta(x) = y \,|\, \theta \in \mathcal{V}_\theta)$. Then, this strategy satisfies

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

where $OPT = \min_{\mathcal{A}} cost(\mathcal{A})$ and π is as in Equation (3.3).

Proof. Through simple calculations, we can see that

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

With this, our result easily follows from the near-optimality result of Equation (3.2) for the general GBS strategy. □

3.3 Kernel Classifiers

With the above, we turn to the problem of efficiently realizing 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*,[1] 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 $\mathcal{F}$:

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

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

One major problem of these classifiers is the high-dimensionality of $\mathcal{F}$, being even infinite-dimensional in certain cases [Mohri et al., 2012]. However, it is always possible to restrict the set of parameters to a finite-dimensional subspace of $\mathcal{F}$ without loss, which is the subject of our next theorem:

[1] 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 $\lambda > 0$

Lemma 3.6. *Let $S = span(\phi(x_1), \dots, \phi(x_N))$ and let $f|_S$ denote 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 $(b, f) \in \mathbb{R} \times \mathcal{F}$.

Proof. 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 complement of S. In particular, this guarantees that any $f \in \mathcal{F}$ can be decomposed 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. □

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

Definition 3.7 (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 set of parameters.

Now, we wish to apply the GBS strategy over 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

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

with K_i being the i-th row of K. With this, the cut probabilities are given by (c.f. Theorem 3.5)

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

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 $p_{x_j,\pm}$.

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 $|\mathcal{L}|+2$, a more tractable range of values.

We start with a simplifying assumption:

Assumption for this section: In what follows, we assume the labeled set $\mathcal{L}$ to be composed of the first t data points $\{x_1, \ldots, x_t\}$, which is always possible to do since it amounts to a simple re-indexing of data points. Besides, we assume a uniform prior $p_0(b, \alpha)$ over the set of parameters 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 cut probabilities 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. Besides, 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

$$\mathcal{V}_t = \{(b, \beta) \in \Omega_\beta : y_i(b + \beta^T L_i) \geq 0,\ 1 \leq i \leq t\} \tag{3.4}$$

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

$$p_{x_j,\pm} = \mathbb{P}\left(\pm(b' + \beta'^T L_j) > 0\right) \tag{3.5}$$

Due to its length, the proof will be left to Appendix A. 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 an N-dimensional space, and thus can be costly to compute. To fix this, we first construct an orthonormal basis for S_j:

Lemma 3.9. *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 $T_j = (\underbrace{0, \ldots, 0}_{t+1}, L_j^{t+1}, \ldots, L_j^N)$.

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

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

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

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

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

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

and then normalizing, which concludes our demonstration.

□

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

$$(b', \beta') = \sum_{i=1}^{t+1} w_i e_i + w_{t+2} \frac{T_j}{\|T_j\|} \tag{3.6}$$

and $w \in \mathbb{R}^{t+2}$ contains the orthonormal basis coefficients. By replacing this representation in Equations (3.4) and (3.5), we obtain our main result:

Definition 3.10 (Partial Cholesky Factor). *Let $\mathcal{L} = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ 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*

$$\ell_i = \left(1, L_i^1, \ldots, L_i^t, \sqrt{K_{ii} - \sum_{k=1}^{t} (L_i^k)^2}\right) \tag{3.7}$$

In particular, we note that $\ell_i^r = 0$ whenever $i \leq \min(t, r-2)$.

Theorem 3.11. *Let w be a random vector drawn uniformly over the set*

$$W_t = \{w \in \mathbb{R}^{t+2} : \|w\| \leq 1 \text{ and } y_i w^T \ell_i \geq 0,\ 1 \leq i \leq t\} \tag{3.8}$$

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

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

Proof. By Lemma 3.8, the cut probabilities satisfy $p_{x_j,+} = \mathbb{P}(b' + \beta'^T L_j > 0)$, where (b', β') is uniformly distributed over $\mathcal{V}_t \cap S_j$. However, (b', β') can also be written in terms of the orthonormal basis coefficients of Equation (3.6), resulting in

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

In particular, this implies that sampling $(b', \beta') \sim Unif(S_j \cap \mathcal{V}_t)$ is equivalent to sampling $w \sim Unif(W_t)$. Finally, we perform the same substitution on the cut probabilities and 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

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

□

In conclusion, we have shown that the cut probabilities $p_{x_j,\pm}$ only 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 main advantages compared to previous works:

- *Low Memory Footprint*: ALuMA [Gonen et al., 2013] has to compute the entire $N \times N$ Cholesky decomposition matrix while our method only needs the first t columns. This truncated computation is very similar to the

Incomplete Cholesky Decomposition techniques [Smola and Schökopf, 2000, Bach and Jordan, 2005] but without a dedicated pivot-selection strategy. In fact, our method does not aim to compute a low-rank approximation of the kernel matrix K but only to provide the means for efficiently sampling the version space.

- *Efficient Sampling*: Unlike KQBC [Gilad-Bachrach et al., 2006], the version space sample w is independent of the particular data point x_j, translating into a more efficient estimation of the cut probabilities $p_{x,\pm}$ over large sets of data points.
- *Optimizations*: The Cholesky property allows us to introduce a novel sampling optimization called *rounding cache*, which significantly speeds up our sampling procedure. Refer to Section 4.2.1 for more details.

A more complete discussion on the above points will be left to Chapter 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. On 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 $\mathcal{X}$, especially 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 3.12. *Let $\mathcal{X}$ be a data set, and let h^* be the target classifier. Then, we define the* ***accuracy error*** *of a classifier h over $S \subset \mathcal{X}$ to be*

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

Lemma 3.13. *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.

Proof. Our objective is to apply Theorem 2.2 from Mohri et al. [2012], which is given as follows: 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 $[h]$. □

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 can compute an increasingly more accurate classifier h over S. In particular, we note that choosing a fast converging AL algorithm lets 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 given 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 in many cases. For example, if $\mathcal{H}$ has finite VC-dimension D then by Sauer's Lemma [Sauer, 1972] $|\hat{\mathcal{H}}| = \mathcal{O}(|\mathcal{X}|^D)$, resulting in a logarithmic dependency of m on the data set size.

In the interactive model development scenario, it is often the case that data sets suffer from high class imbalance, especially in data exploration tasks. For such cases, it would be ideal to also obtain generalization bounds for performance metrics other than accuracy, such as precision, recall, and F-score. This is explored in our next result:

Definition 3.14 (Precision, Recall, and F-score errors). *For any $r, s \in \{-, +\}$, let us 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

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

Theorem 3.15. *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.

Proof. 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. □

By combining the above result with Theorems 3.13, it is possible to achieve similar error guarantees in precision, recall, and F-score by having a sample $\frac{1}{\mu^2}$-times larger. However, whether this extra factor is acceptable in practice will depend on the use case. For example, data exploration tasks tend to be very selective, with the user interest usually covering $\mu = 1\%$ or less of the entire database, which corresponds to an increase in sample size by a large factor of 10,000 or more. Refer to Appendix D for some concrete examples of data exploration tasks and their selectivity factors.

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 labels are known. However, the lack of 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 an 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 3.16 (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*

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

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

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 (3.1).

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

Theorem 3.17. *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 3.13, with probability $1 - \delta$ it holds

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

Proof. 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. □

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

3.5.2 Approximations for Majority Vote

Despite its good theoretical properties, the Majority Vote classifier cannot be readily used in practice because the cut probabilities $p^{\pi}_{x,y}$ cannot be exactly computed. However, those probabilities can usually be approximated by relying on

the Law of Large Numbers or equivalent statements, which leads us to the following definition:

Definition 3.18 (Approximated Majority Vote). *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 approximated majority vote classifier is defined as*

$$\hat{MV}^{\pi}(x) = \arg\max_{y \in \mathcal{Y}} \hat{p}^{\pi}_{x,y} \tag{3.9}$$

where $\hat{p}^{\pi}_{x,y} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left(H_i(x) = y\right)$ and $\{H_i\}_{i=1}^{M}$ is an i.i.d sample from π.

In resume, $\hat{MV}^{\pi}$ approximates MV^{π} through a sample of classifiers $\{H_i\}_{i=1}^{M}$. In fact, by the Law of Large Numbers, this approximation should have a similar performance as the original Majority Vote classifier, as shown in our next result:

Theorem 3.19. *Under the conditions of Lemma 3.13, the random variable:*

$$C^{r,s}(\hat{MV}^{\pi}, S) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\left(\hat{MV}^{\pi}(X_i) = r, h^*(X_i) = s\right) \tag{3.10}$$

converges in probability to $C^{r,s}(MV^{\pi}, \mathcal{X})$ as $m, M \to \infty$. In particular, this result also holds for the Accuracy, Precision, Recall, and F-Score error metrics.

Proof. In what follows, we focus on showing that $C^{r,s}(\hat{MV}^{\pi}, S) \to C^{r,s}(MV^{\pi}, \mathcal{X})$ in probability in the joint limit $m, M \to \infty$, the remainder of the proof being a simple application of Slutsky's theorem.

First, we simplify the notation by droping π from MV^{π}, and we also define X to be a uniformly random sample from $\mathcal{X}$. Now, given any $\epsilon > 0$ we have:

$$\begin{aligned}
&\mathbb{P}\left(\left|C^{r,s}(\hat{MV}, S) - C^{r,s}(MV, \mathcal{X})\right| > 2\epsilon\right) \leq \\
&\quad \mathbb{P}\left(\left|C^{r,s}(\hat{MV}, S) - C^{r,s}(MV, S)\right| > \epsilon\right) + \mathbb{P}\left(|C^{r,s}(MV, S) - C^{r,s}(MV, \mathcal{X})| > \epsilon\right)
\end{aligned}$$

In particular, it is enough to show that each term above converges to zero as

$m, M \to \infty$. First, by a simple application of Markov's inequality we have:

$$
\begin{aligned}
&\mathbb{P}\left(\left|C^{r,s}(\hat{MV}, S) - C^{r,s}(MV, S)\right| > \epsilon\right) \\
&= \mathbb{P}\left(\frac{1}{m}\left|\sum_{i=1}^{m} \mathbb{1}(\hat{MV}(X_i) = r, h^*(X_i) = s) - \mathbb{1}(MV(X_i) = r, h^*(X_i) = s)\right| > \epsilon\right) \\
&\leq \mathbb{P}\left(\frac{1}{m}\sum_{i=1}^{m}\left|\mathbb{1}(\hat{MV}(X_i) = r, h^*(X_i) = s) - \mathbb{1}(MV(X_i) = r, h^*(X_i) = s)\right| > \epsilon\right) \\
&\leq \mathbb{P}\left(\frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(\hat{MV}(X_i) \neq MV(X_i), h^*(X_i) = s) > \epsilon\right) \\
&\leq \frac{1}{\epsilon}\mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(\hat{MV}(X_i) \neq MV(X_i), h^*(X_i) = s)\right] \\
&= \frac{1}{\epsilon}\mathbb{P}\left(\hat{MV}(X) \neq MV(X), h^*(X) = s\right) \\
&\leq \frac{1}{\epsilon}\mathbb{P}\left(\hat{MV}(X) \neq MV(X)\right) \\
&= \frac{1}{\epsilon}\mathbb{P}\left(\hat{p}_{X,MV(X)} < 0.5\right) \\
&= \frac{1}{\epsilon|\mathcal{X}|}\sum_{x\in\mathcal{X}} \mathbb{P}\left(\hat{p}_{x,MV(x)} < 0.5\right)
\end{aligned}
$$

Finally, the Law of Large Numbers guarantees that $\hat{p}_{x,y} \to p_{x,y}$ almost surely for any (x, y) as $M \to \infty$ which, in particular, shows that $\hat{p}_{x,MV(x)} \to 1$ for all $x \in \mathcal{X}$ and the above term must converge to zero.

Next, we consider the right term. Since $\mathbb{E}[C^{r,s}(MV, S)] = C^{r,s}(MV, \mathcal{X})$, Chebyshev's inequality gives:

$$
\begin{aligned}
\mathbb{P}\left(|C^{r,s}(MV, S) - C^{r,s}(MV, \mathcal{X})| > \epsilon\right) &\leq \frac{1}{\epsilon^2} Var\left[C^{r,s}(MV, S)\right] \\
&\leq \frac{1}{\epsilon^2 m} Var\left[\mathbb{1}\left(\hat{MV}^{\pi}(X) = r, h^*(X) = s\right)\right] \\
&\leq \frac{1}{\epsilon^2 m}
\end{aligned}
$$

which converges to zero as $m \to \infty$, and concludes the proof. □

In resume, the above theorem not only guarantees that $\hat{MV}$ becomes an increasingly more accurate approximation of MV as $M \to \infty$, but it also guarantees that an accurate $\hat{MV}$ over a large enough sample is also accurate over the entire dataset.

In the following theorem, we develop an asymptotic lower bound for the classification accuracy of MV over in the active learning case. In particular, the same result holds for the approximated majority vote:

Theorem 3.20. *Let $S \subset \mathcal{X}$, and define $S^{\pm}_{\epsilon} = \{x \in S : p_{x,\pm} > 1 - \epsilon\}$ for some $\epsilon \in (0, \frac{1}{2}]$. Then, as the version space approaches a single element, we have that $\epsilon_{fscore}(MV^{\pi}, S)$ is lower bounded by the harmonic mean of $|S^{+}_{\epsilon}|/|S^{+}_{1/2}|$ and $|S^{+}_{\epsilon}|/(|S| - |S^{-}_{\epsilon}|)$.*

Proof. Define $Q^{\pm}_S = \{x \in S : h^*(x) = \pm\}$ and $\delta^{\pm}_{\epsilon} = |S^{+}_{\epsilon} \backslash (Q^{\pm}_S \cap S^{\pm}_{1/2})|$. It is easy to see that $S^{\pm}_{\epsilon} \subseteq S^{\pm}_{1/2}$ and, consequently, $|Q^{\pm}_S \cap S^{\pm}_{1/2}| \geq |Q^{\pm}_S \cap S^{\pm}_{1/2} \cap S^{\pm}_{\epsilon}| = |S^{\pm}_{\epsilon}| - \delta^{\pm}_{\epsilon}$. With this, we can easily create the following bounds for both precision and recall:

$$precision = \frac{|Q^{+}_S \cap S^{+}_{1/2}|}{|S^{+}_{1/2}|} \geq \frac{|S^{+}_{\epsilon}| - \delta^{+}_{\epsilon}}{|S^{+}_{1/2}|}$$

$$recall = \frac{|Q^{+}_S \cap S^{+}_{1/2}|}{|Q^{+}_S|} \geq \frac{|Q^{+}_S \cap S^{+}_{1/2}|}{|S| - |Q^{-}_S \cap S^{-}_{1/2}|} \geq \frac{|S^{+}_{\epsilon}| - \delta^{+}_{\epsilon}}{|S| - (|S^{-}_{\epsilon}| - \delta^{-}_{\epsilon})}$$

Therefore, the *F-score*, which is a harmonic mean of the *precision* and *recall*, is lower bounded by the harmonic mean of the above lower bounds. However, as the number of labeling iterations T increases, the version space V gets smaller and smaller and, consequently, $p_{x,+}$ approaches $\mathbb{1}(h^*(x) = 1)$ for all $x \in S$. As a result, we have $\delta^{\pm}_{\epsilon} = |S^{\pm}_{\epsilon} \setminus (Q^{\pm}_S \cap S^{\pm}_{1/2})|$ converge to zero as T increases. This concludes the proof. □

3.6 Chapter Summary

In this chapter, we have developed a new theoretical framework for efficiently realizing the Generalized Binary Search (GBS) strategy over kernel classifiers. Compared to similar works in the literature [Gilad-Bachrach et al., 2006, Gonen et al., 2013], our technique possesses several advantages. On one hand, compared to KQBC [Gilad-Bachrach et al., 2006], our framework offers strong theoretical guarantees on convergence and an efficient estimation procedure for the cut probabilities that does not require a separate sampling for each data point. On the other hand, compared to ALuMA [Gonen et al., 2013], our technique avoids computing the kernel matrix over all unlabeled points by applying a dimensionality reduction theorem that restricts computations to the set of labeled points, thus greatly enhancing the time and memory cost.

In addition, we have also proved several theoretical bounds on accuracy and F-score which enable our techniques to be applied on a sample of the original data set with minimal performance loss.

Chapter 4

Implementation and Optimizations

In this chapter, we detail how to efficiently implement the results of Theorem 3.11, resulting in an optimized version-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 $p_{x,y}$, which is done via Algorithm 1 below. In what follows, we give a detailed 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 $p_{x_*,y}$, for $x_* \in \mathcal{U}$ and $y = \pm 1$

$K_{ij} \leftarrow k(x_i, x_j)$, $1 \le i, j \le t$
$L \leftarrow Cholesky(K)$
$\ell_i \leftarrow \big(1, L_i^1, \ldots, L_i^t, 0\big)$, $1 \le i \le t$
$W_t \leftarrow \{w \in \mathbb{R}^{t+2} : \|w\| \le 1 \text{ and } y_i \ell_i^T w \ge 0,\ 1 \le i \le t\}$
$\mathcal{E}_{round} \leftarrow rounding\_algorithm(W_t, \mathcal{E}_0)$
$\{w^i\}_{i=1,\ldots,M} \leftarrow hit\_and\_run\,(W_t, M, \mathcal{E}_{round})$
for all $x_* \in \mathcal{U}$ **do**
    $K_* \leftarrow [k(x_1, x_*), \ldots, k(x_t, x_*)]^T$
    $L_* \leftarrow L^{-1} K_*$
    $\ell_* \leftarrow \Big(1, L_*^1, \ldots, L_*^t, \sqrt{k(x_*, x_*) - \|L_*\|^2}\Big)$
    $p_{x_*,y} \leftarrow \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left(y\, \ell_*^T w^i > 0\right)$
end for
return $\{p_{x_*,y},$ for $x_* \in \mathcal{U}$ and $y = \pm 1\}$

4.0.1 Computing the Partial Cholesky Factors ℓ_j

From Theorem 3.11, the first step in estimating $p_{x_j,+}$ is to compute the *partial Cholesky factor* ℓ_j, as defined in Equation (3.7). In particular, this vector can

be easily computed once we have $\tilde{L}_j = (L_j^1, \ldots, L_j^t)$, 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 $\mathcal{O}\left(t^2(d+t) + |\mathcal{U}|t(d+t)\right)$.

4.1 Estimating the Cut Probabilities via Sampling

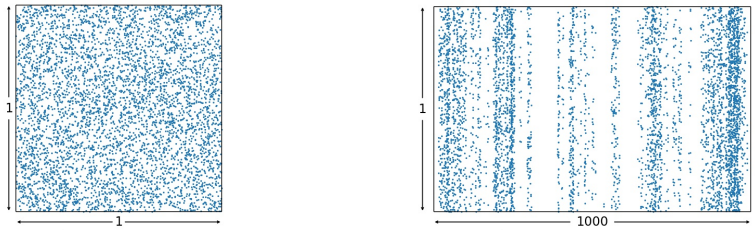
Since the probabilities $p_{x_j,y}$ have no closed-form expression, we estimate these quantities via a *sampling procedure* [Gonen et al., 2013, Gilad-Bachrach et al., 2006]. This step corresponds to lines 6 and 11 in Algorithm 1.

First, by the Law of Large Numbers, we can write

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

where $\{w^i\}_{i=1,\ldots,M}$ is an i.i.d. sample following the uniform distribution over W_t. 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 chain must be generated for each w^i, which can become quite time-consuming in practice. To overcome this overhead, we opt for an optimized sampling procedure that requires generating a single chain. Since the hit-and-run chain $\{w^1, w^2, \ldots\}$ forms a positive Harris-recurrent Markov Chain [Bélisle et al., 1993], the Law of Large



(a) A 1×1 square (b) A 1000×1 rectangle

Figure 4.1: 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.

Numbers theorem for Markov Chains [Kaspi et al., 1997, Theorem 17.1.7] guarantees that

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

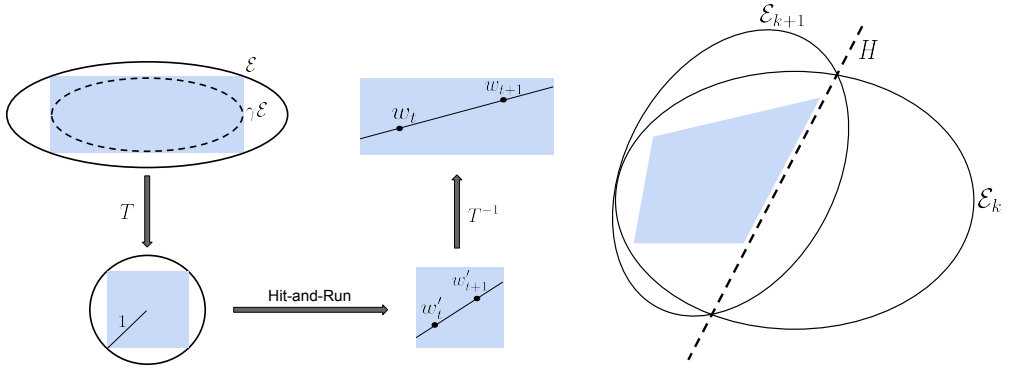
Thus, estimating $p_{x_j,y}$ can be done through a single chain. Furthermore, we employ two well-known techniques for improving the mixing time [Harms and Roebroeck, 2018, Link and Eaton, 2012]: *Burn-in* consists of discounting the first B samples from the chain since their distribution is very far 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 $\{w^B, w^{B+T}, \ldots, w^{B+(M-1)T}\}$ as the final sample.

The technical details surrounding hit-and-run's implementation will be left in 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 $\mathcal{O}(|\mathcal{U}|Mt)$.

4.2 Improving the Sample Quality via Rounding

The hit-and-run algorithm can have a large mixing time, especially when the convex body $W \subset \mathbb{R}^n$ is very elongated in one direction. In practice, this can lead to poor sample quality, whose distribution is far away from uniform even after thousands of steps; see Figure 4.1 for an example of this behavior.

To address the above problem, we adopt a *rounding procedure* [Lovász, 1986, De Martino et al., 2015]. In general terms, rounding is a preprocessing algorithm that attempts to make W more evenly elongated across all directions via



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

a linear transformation T. Once this 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 4.2(a) for an illustration of this process.

One well-known method for computing a rounding transformation is via Lovász's 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.[1] The main idea is to find a *rounding ellipsoid* $\mathcal{E}$ satisfying

$$\gamma\mathcal{E} \subset W \subset \mathcal{E} \tag{4.1}$$

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, $\mathcal{E}$ should approximately capture the directions of major elongation of W; thus, by choosing T as any linear transformation mapping $\mathcal{E}$ into a unit ball, any major stretching of W should also be corrected, and the hit-and-run chain should mix quickly. More precisely, the resulting hit-and-run chain is guaranteed to mix in $\mathcal{O}^*(n^2/\gamma^2)$ steps.

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.

Algorithm 2 presents an outline of the ellipsoid computation algorithm, with a complete description deferred to Appendix C. Starting from any ellipsoid $\mathcal{E}_0 \supset W$,

[1] The scaling matrix must be symmetric and positive-definite.

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

```
k ← 0
while γ𝓔_k ⊄ W do
    H ← get_separating_halfspace(𝓔_k, W)
    𝓔_{k+1} ← minimum_volume_ellipsoid(𝓔_k, H)
    k ← k + 1
end while
return 𝓔_k
```

the algorithm constructs a sequence $\{\mathcal{E}_1, \mathcal{E}_2, \ldots\}$ that gets step by step closer to $\mathcal{E}^*$. More precisely, given $\mathcal{E}_k \supset W$, it proceeds as follows:

1. First, the algorithm checks whether $\gamma\mathcal{E}_k \subset W$ for a given threshold $0 < \gamma < 1/n$. If so, $\mathcal{E}_k$ is returned as a 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
$$\alpha(\mathcal{E}, H) = \frac{a^T z - b}{\sqrt{a^T P a}}$$
See Figure 4.2(b) for an illustration.
3. Finally, we set $\mathcal{E}_{k+1}$ as the *minimum volume ellipsoid* containing $\mathcal{E}_k \cap H(a, b)$, which can be analytically computed via Algorithm 8.

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. When put together, these factors ensure an upper bound of

$$\frac{\log\left(vol(\mathcal{E}_0)/vol(W)\right)}{\log\left(1/c\right)} \tag{4.2}$$

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, especially as the dimensionality increases. This is especially 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, to improve its efficiency, we propose two major optimizations:

1. *Ellipsoid caching* (Section 4.2.1): 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 $\mathcal{E}_0$ of small volume. In other words, $\mathcal{E}_0$ is used as a *warm-start* for the ellipsoid algorithm.

2. *Optimized ellipsoid algorithm* (Section 4.2.2): 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.

4.2.1 The Ellipsoid Caching Algorithm

To tackle the high number of rounding iterations, we introduce an *ellipsoid caching* procedure. Based on Equation (4.2), one way of reducing the number of iterations is to find an initial ellipsoid $\mathcal{E}_0$ of volume as small as possible to warm-start the rounding algorithm. 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 also be close in size, which gives us hope in constructing $\mathcal{E}_0$ of volume close to W_{t+1} as well.

First, let us 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 (3.8), t being 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

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 4.1. *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.

Proof. Let us consider an ellipsoid of 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, resulting in

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

which concludes our proof. □

4.2.2 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. To avoid these numerical issues, we introduce a new ellipsoid computation algorithm for convex bodies of 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 $\alpha(\mathcal{E}, H) > -\gamma$. To tackle the first problem, we rely on the following result:

Lemma 4.2. *Consider the convex body* $W = \{w \in \mathbb{R}^n : \|w\| \leq 1 \text{ and } a_i^T w \leq 0,\ 1 \leq i \leq m\}$, *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* $\alpha_w = \alpha(\mathcal{E}, H(w, 1)) = (z^T w - 1)/\sqrt{w^T P w}$.

Proof. 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. □

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 (e.g. SLSQP, COBYLA). 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 $\gamma\mathcal{E} \subset W \subset \mathcal{E}$

$k \leftarrow 0$
while True **do**
  $\alpha_{i^*} \leftarrow \max_i \frac{z_k^T a_i}{\sqrt{a_i^T P_k a_i}}$
  $\alpha_z \leftarrow \frac{\|z_k\|(\|z_k\|-1)}{\sqrt{z_k^T P_k z_k}}$
  if $\alpha_{i^*} \geq \alpha_z$ and $\alpha_{i^*} > -\gamma$ **then**
    $\mathcal{E}_{k+1} \leftarrow minimum\_volume\_ellipsoid(\mathcal{E}, H(a_{i^*}, 0))$
  else if $\alpha_z \geq \alpha_{i^*}$ and $\alpha_z > -\gamma$ **then**
    $\mathcal{E}_{k+1} \leftarrow minimum\_volume\_ellipsoid(\mathcal{E}, H(z_k, \|z_k\|))$
  else
    $\alpha_{w^*} \leftarrow \max_{\|w\|=1} \frac{w^T z_k - 1}{\sqrt{w^T P_k w}}$
    if $\alpha_{w^*} \leq -\gamma$ **then**
      return $\mathcal{E}_k$
    end if
    $\mathcal{E}_{k+1} \leftarrow minimum\_volume\_ellipsoid(\mathcal{E}_k, H(w^*, 1))$
  end if
  $k \leftarrow k + 1$
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 when $z \notin W$.

Finally, by putting together all of the above considerations, we are now ready to present our optimized rounding algorithm for W, which is 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 $a^T P a \leq 0$. Similarly, this problem is also 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 use 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; refer to Appendix C for

the complete implementation details. 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 removes the diagonalization step and eliminates the dependency on P via the decomposition, thus solving the numerical instability issue.

4.3 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 $\gamma > 0$, it also guarantees that $z \in W_t$, which can be used as a starting point for hit-and-run.

4.4 Experimental Results

In this section, we evaluate our OptVS algorithm against state-of-the-art active learners [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 slow start problem. All algorithms are part of our Python-based prototype.[2]

Experimental Setup

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

(1) Sloan Digital Sky Survey (SDSS, 190 million points): The data set contains the "PhotoObjAll" table with 510 attributes and 190 million sky observations.[3] We used a 1% sample (1.9 million points, 4.9GB) to create a data set for running active learning algorithms—our formal results 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.

[2] Our code is available at `https://gitlab.inria.fr/aideme/aideme`
[3] `http://www.sdss3.org/dr8/`

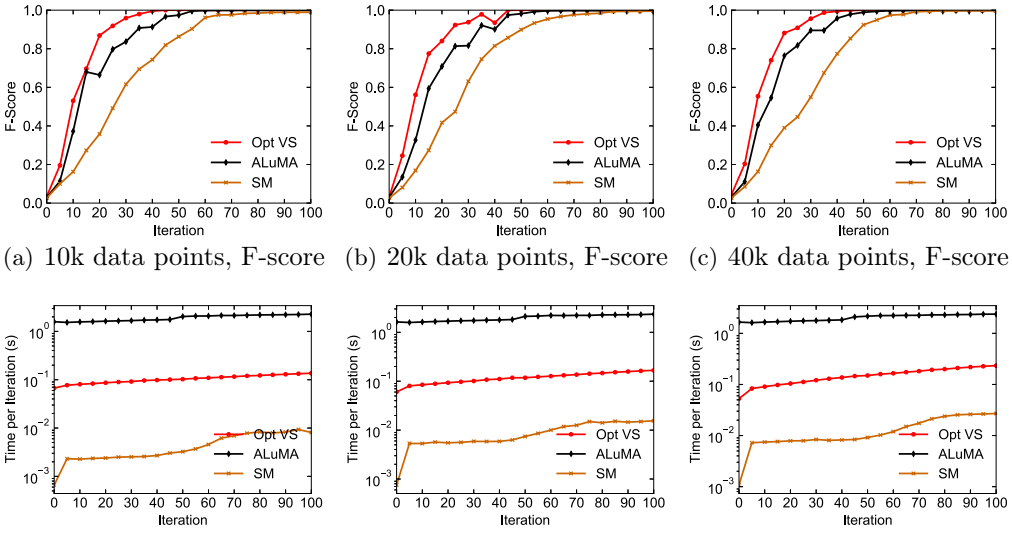
SDSS also offers a query release where the SQL queries reflect the data interest of scientists.[4] We extracted 11 queries to build a benchmark, as shown in Appendix D, and treated them as the ground truth of the positive classes of 11 classifiers to be learned—our system does *not* need to know these queries in advance but can learn them via active learning. These queries reflect the 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 a 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. These queries include 4–10 attributes, with a mix of 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]. We also compare to the unfactorized DSM [Huang et al., 2018] over low-dimensional SDSS queries only, being too expensive to run in the remaining cases. 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 and negative example chosen at random and runs up to 200 additional labeled examples (iterations). At every iteration, the classification accuracy of each active learner is computed over the entire data set used for training, which corresponds to the user's interest metric for practical use cases like data exploration and annotation. Additionally, each experiment was repeated 10 times, and individual results were averaged.

Hyper-parameter tuning: In the Active Learning scenario, hyper-parameter tuning is a much more complex problem than in traditional ML settings, especially in the data exploration and annotation applications. More precisely, common model selection strategies, like cross-validation, cannot be easily applied since no large, independent set of labeled examples is available for tuning. Because of this, we have decided to employ a simplified hyper-parameter selection procedure: for each algorithm, we choose the hyper-parameters that achieve the highest accuracy over the entire query population (SDSS and Car queries) while running under interactive performance (or close to). In doing so, the hyper-parameters computed can be interpreted as "reasonable default values" for each algorithm, especially over low selectivity use cases.

[4] http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp



(a) 10k data points, F-score (b) 20k data points, F-score (c) 40k data points, F-score

(d) 10k data points, Time (e) 20k data points, Time (f) 40k data points, Time

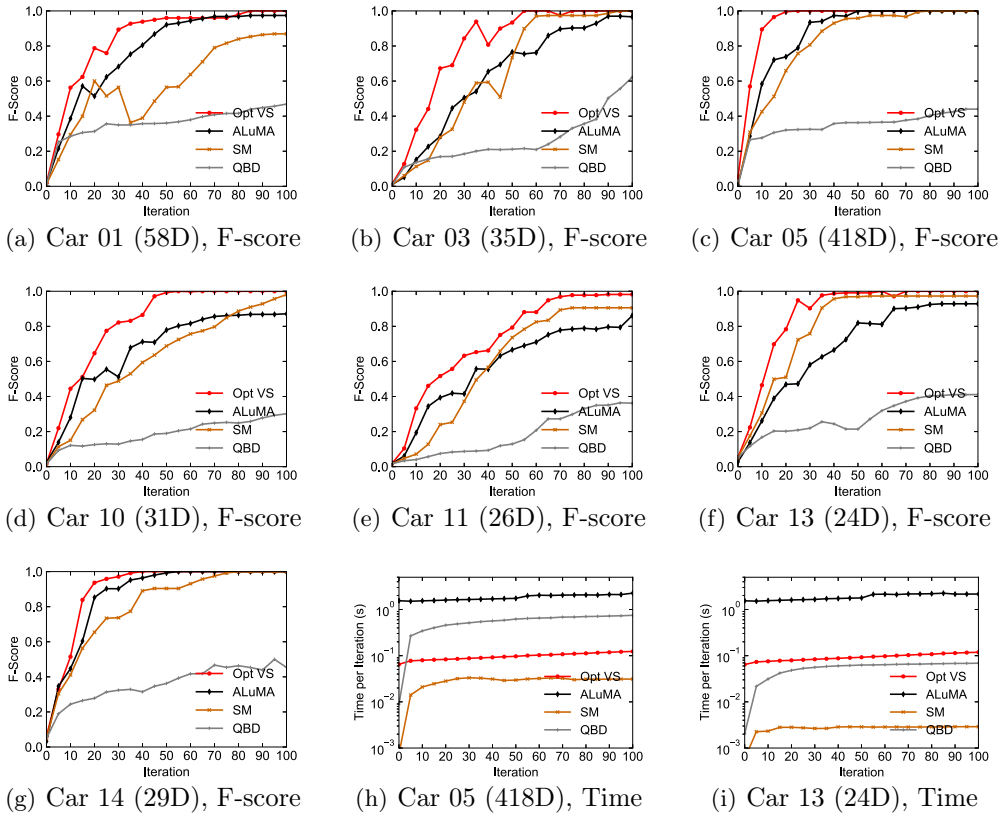
Figure 4.3: Comparison of OptVS with ALuMA [Gonen et al., 2013] and Simple Margin [Tong and Koller, 2001] in terms of classification accuracy and time per iteration using a synthetic data set of various sizes.

Next, we give a detailed description of the hyper-parameters and other configuration values used by each algorithm in our experiments. All active learners use an RBF kernel with a default scaling of 1/num_features, except for the SDSS 04 for which we used 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 weighted by 10^{-5} during training. As for ALuMA, 16 hypotheses are sampled from the version space via independent hit-and-run chains of length 2000. Additionally, our OptVS algorithm samples 25 hypotheses from a single hit-and-run chain, using a 1000 steps burn-in and 100 steps thinning. 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 $\gamma = 1/(n+1)\sqrt{n}$, n being the dimension of W_t.

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.

Comparison to ALuMA

One major limitation of the ALuMA [Gonen et al., 2013] algorithm is its costly preprocessing step: to support kernels, it requires computing the kernel matrix over the entire data set before applying a dimensionality reduction technique, a



(a) Car 01 (58D), F-score (b) Car 03 (35D), F-score (c) Car 05 (418D), F-score

(d) Car 10 (31D), F-score (e) Car 11 (26D), F-score (f) Car 13 (24D), F-score

(g) Car 14 (29D), F-score (h) Car 05 (418D), Time (i) Car 13 (24D), Time

Figure 4.4: Comparison of OptVS with other active learning techniques [Tong and Koller, 2001, Settles, 2012, Gonen et al., 2013] in terms of classification accuracy and time per iteration using the Car data set. In addition, we display in parenthesis the dimensionality of each query after one-hot encoding.

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.

To enable a more complete comparison with ALuMA, additionally to the Car user patterns we also generate a synthetic data set corresponding to a subsampled version of SDSS 02. More precisely, our synthetic data is composed of $N \in \{1 \times 10^4, 2 \times 10^4, 4 \times 10^4\}$ data points sampled uniformly over $[-\sqrt{3}, \sqrt{3}]^2$, with the user interest region being a centered ball containing 1% of the data. In all considered cases, after applying the ALuMA preprocessing, the transformed data set has a dimensionality close to 300. We also note that while ALuMA is run over this transformed data set, all other algorithms use the original data.

We start by observing the classification accuracy results of Figures 4.3 and 4.4. In all considered cases, OptVS outperforms ALuMA at every iteration.

Query	OptVS			ALuMA			SM			QBD		
	10	20	50	10	20	50	10	20	50	10	20	50
01	**56**	**79**	**96**	39	51	92	29	60	56	29	31	36
02	**61**	**91**	95	33	64	91	18	57	**97**	9	13	21
03	**32**	**67**	**93**	15	29	77	11	28	73	14	17	21
04	**33**	**48**	**81**	15	29	51	9	27	65	7	8	17
05	**89**	**100**	**100**	58	74	**100**	43	66	96	28	32	36
06	**90**	**100**	**100**	80	**100**	**100**	73	**100**	**100**	47	48	66
07	**72**	**77**	**93**	40	59	72	38	56	87	21	32	45
08	**100**	**100**	**100**	**99**	**100**	**100**	**100**	**100**	**100**	56	67	93
09	**76**	98	**100**	56	74	**100**	71	**99**	**100**	38	39	63
10	**44**	**65**	**99**	28	50	78	15	32	69	12	13	19
11	**33**	**52**	**79**	19	39	67	7	24	74	4	7	13
12	**51**	**80**	**99**	10	25	61	16	58	95	6	9	53
13	**46**	**78**	**99**	26	47	82	31	51	97	17	20	21
14	**51**	**94**	**100**	45	85	99	41	65	90	24	28	36
15	38	**73**	**97**	32	64	87	36	66	93	**41**	46	42
16	**86**	**100**	**100**	40	76	**100**	53	78	**100**	29	34	52
17	**27**	**37**	**53**	25	28	39	11	22	47	10	12	14
18	**94**	91	**100**	69	87	98	82	**96**	**100**	38	50	69

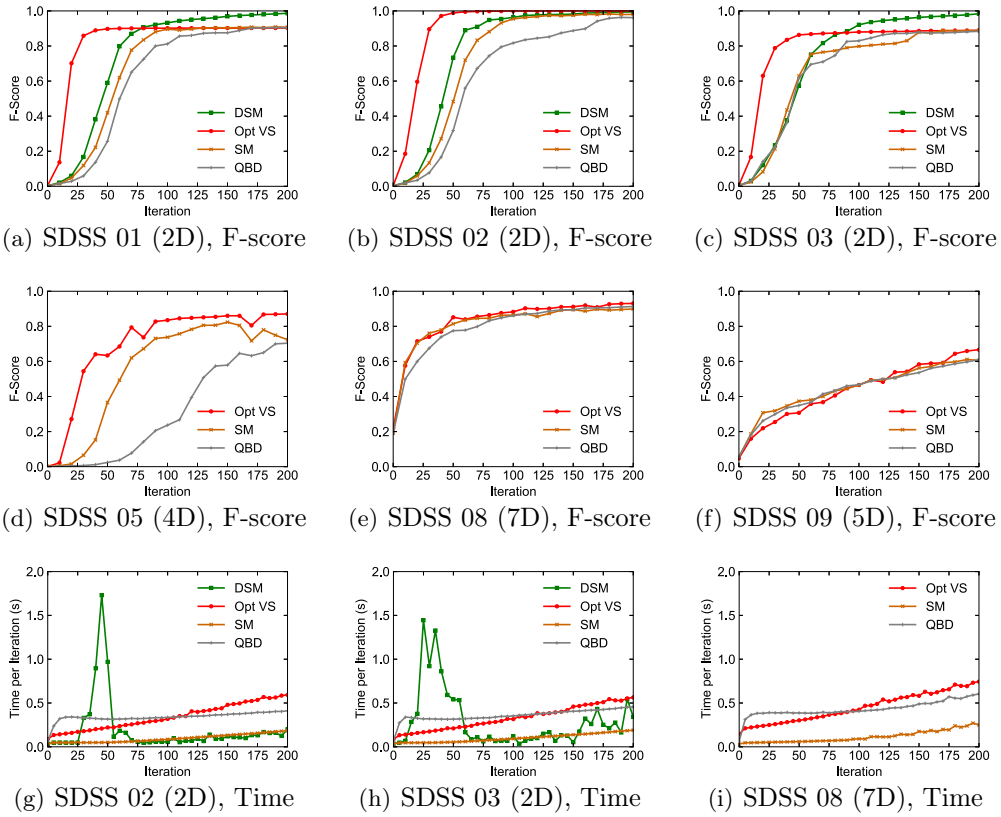
Table 4.1: F-score (in %) comparison at iterations 10, 20, and 50 of OptVS with other active learning techniques [Tong and Koller, 2001, Settles, 2012, Gonen et al., 2013] using the Car data set. Bold values represent the observed maximum across all algorithms per query at the given iteration.

Figures 4.4(b) and 4.4(d) show the results for two particular Car queries. In each case, our algorithm manages to reach perfect accuracy within nearly 50 labeling iterations, with ALuMA still being at 80% and all other algorithms at 70% or below. Additionally, Table 4.1 shows the complete set of results over all Car queries; over there, we can see that OptVS offers a much faster convergence speed than ALuMA in earlier iterations, being, on average, 20 F-score points higher across iterations 10 and 20.

Next, we observe how the algorithms compare in terms of time per iteration. Plots 4.3(d)–4.3(f) display the time measurements over the synthetic queries, while Figures 4.4(h)–4.4(i) contain the two most expensive Car patterns. In all cases, ALuMA is by far the most expensive algorithm, being around 10 times slower than OptVS. We also note that Simple Margin is the fastest algorithm due to its efficient margin-based selection strategy.

In summary, we can draw the following conclusions:

- *ALuMA preprocessing*: ALuMA's preprocessing step is very memory intensive and could not be run beyond 40k data points. In contrast, OptVS has no such restrictions.



(a) SDSS 01 (2D), F-score (b) SDSS 02 (2D), F-score (c) SDSS 03 (2D), F-score

(d) SDSS 05 (4D), F-score (e) SDSS 08 (7D), F-score (f) SDSS 09 (5D), F-score

(g) SDSS 02 (2D), Time (h) SDSS 03 (2D), Time (i) SDSS 08 (7D), Time

Figure 4.5: Comparison of OptVS with other active learning techniques [Tong and Koller, 2001, Settles, 2012, Huang et al., 2018] in terms of classification accuracy and time per iteration using the SDSS data set. In addition, we display in parenthesis the dimensionality of each query.

- *Performance*: In all data sets considered, OptVS outperforms ALuMA at every iteration; for instance, over Car queries, we observed an improvement of 20 F-score points in average over the initial iterations. Despite employing similar exploration strategies, the difference in performance can be explained by two main factors: the use of *rounding* (Section 4.2) to improve the version space sample quality and the high-dimensionality of data after ALuMA's preprocessing (around 300).

- *Efficiency*: In all observed cases, our OptVS algorithm runs around 0.1 seconds per iteration at all times, while ALuMA takes close to 1 second. The performance benefits of our algorithm come mainly from the single-chain sampling optimization of Section 4.1.

Due to the above, we will leave ALuMA out of consideration in the remaining experimental evaluations.

Query	OptVS			DSM			SM			QBD		
	25	50	100	25	50	100	25	50	100	25	50	100
01	**81**	**90**	90	10	59	**93**	8	42	90	4	26	81
02	**83**	**99**	**100**	11	73	96	9	48	95	5	32	82
03	**67**	**86**	88	19	57	**92**	15	63	80	18	62	83
04	**97**	**100**	**100**	26	80	98	86	98	**100**	91	96	97
05	**38**	**63**	**84**	–	–	–	3	37	74	0	2	24
06	3	4	7	–	–	–	**5**	**7**	**10**	4	4	5
07	40	44	47	–	–	–	**44**	**48**	51	41	46	**54**
08	**80**	**85**	**88**	–	–	–	70	81	86	64	78	86
09	21	31	**47**	–	–	–	**32**	**37**	**47**	27	35	**47**
10	32	41	**62**	–	–	–	**34**	**50**	59	29	39	53
11	**43**	50	51	–	–	–	**43**	**51**	57	34	45	**59**

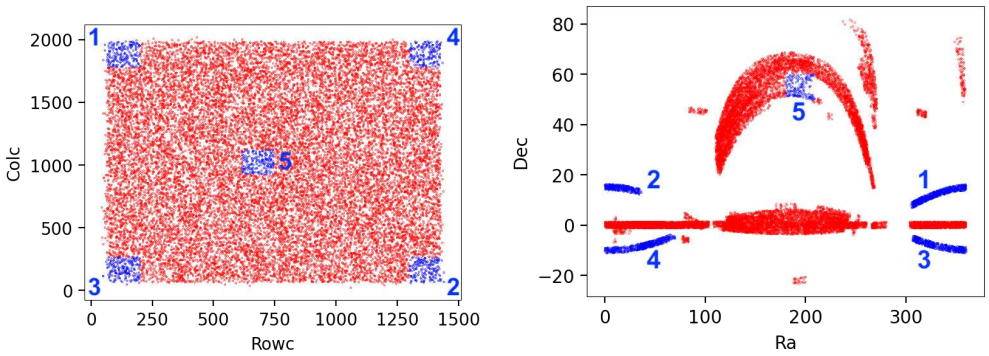
Table 4.2: F-score (in %) comparison at iterations 25, 50, and 100 of OptVS with other active learning techniques [Huang et al., 2018, Tong and Koller, 2001, Settles, 2012] using the SDSS data set. Bold values represent the observed maximum across all algorithms per query at the given iteration. DSM results are limited due to its high time cost.

Comparing with Other Techniques

Next, we compare our algorithm to two VS algorithms, Simple Margin (SM) [Tong and Koller, 2001] and Query-by-Disagreement (QBD) [Settles, 2012], as well as unfactorized DSM [Huang et al., 2018]. Figures 4.4 and 4.5 illustrate the per-iteration measurements for selected SDSS and Car queries, with Tables 4.2 and 4.1 showing a complete set of results. Comparison with unfactorized DSM is restricted to the low-dimensional SDSS queries 01–04 due to the high time cost of computing convex hulls, even in moderately high dimensions.

Let's first consider the results of the SDSS data set. Our OptVS algorithm outperforms the two VS algorithms across the majority of queries, including DSM most time over SDSS 01–04. For example, let's consider Figures 4.5(b) and 4.5(c) showing the results for SDSS 02 and 03. During the initial iterations, OptVS improves much faster than other algorithms and remains the best across all iterations of 02 and in most iterations of 03. We also observe that DSM eventually surpasses kernel-based methods such as OptVS for rectangular patterns such as SDSS 01 and 03. This difference comes from the different classification models being used: DSM's polytope model can easily capture such polygonal patterns, while the round-shaped RBF kernel will have trouble learning the corners.

We can also observe similar results over the Car user patterns. Once again, OptVS tends to achieve a faster convergence speed in initial iterations when compared to other VS methods; in particular, we note that OptVS managed to reach $\geq 95\%$ accuracy for 14 out of 18 queries while labeling less than 50 data



(a) SDSS C1 (2D, 5%) (b) SDSS C2 (2D, 13%)

Figure 4.6: Labeled data distribution for two synthetic patterns, SDSS C1 and SDSS C2, whose user interest regions are composed of multiple disjoint clusters. In parenthesis, we display the dimensionality and the selectivity of each query.

points. As mentioned before, Car queries present faster convergence rates than SDSS due to the data sparsity around the decision boundary, making it easier for algorithms to separate positive and negative classes.

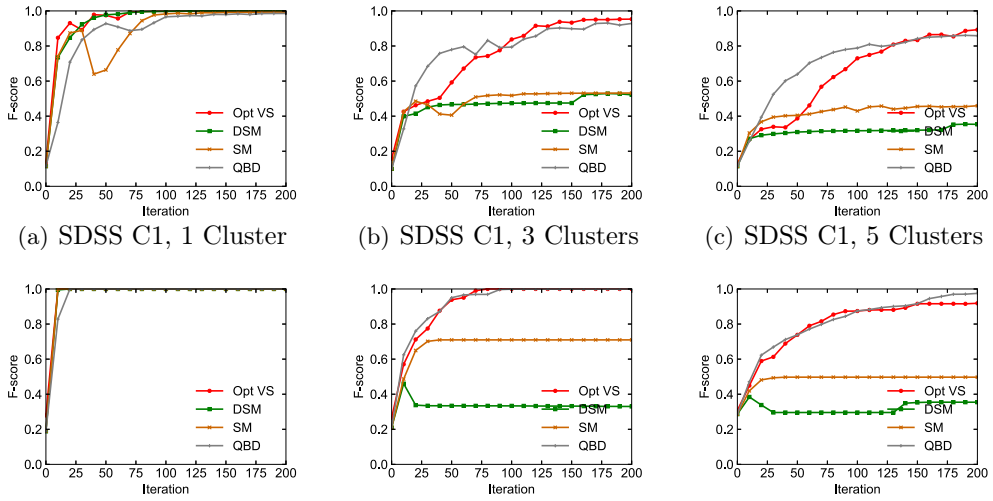
In terms of running time, all algorithms run under interactive performance taking at most a couple of seconds per iteration. In particular, we note that our OptVS algorithm is very fast, taking less than 1 second per iteration even when training over hundreds of labeled points. In particular, all algorithms except for ALuMA run extremely quickly over the Car data set, taking at most 0.1 seconds per iteration in every case.

Disjoint Interest Patterns

To test the limitations of our VS technique, we created a synthetic scenario where the user interest region is composed of multiple disjoint subsets. The main intuition for this choice comes from the exploration-exploitation principle [Monarch, 2021]: effective active learning strategies must find a balance between *exploiting* the current classification model and *exploring* unseen data regions. Since version space algorithms tend to be more exploitative, we expect them to have trouble learning user patterns composed of multiple disjoint regions.

To check our intuition, we have devised two synthetic labeled sets based on SDSS 01 and 03:

- *SDSS C1*: Displayed in Figure 4.6(a), the user interest is composed of 5 square-shaped clusters of approximately 1% selectivity each.
- *SDSS C2*: Displayed in Figure 4.6(b), this labeled set leverages the natural



(a) SDSS C1, 1 Cluster (b) SDSS C1, 3 Clusters (c) SDSS C1, 5 Clusters

(d) SDSS C2, 1 Cluster (e) SDSS C2, 3 Clusters (f) SDSS C2, 5 Clusters

Figure 4.7: Study of the impact on performance of version space technique with an increasing number of disjoint positive regions. The tag "n Clusters" for each plot means that the user interest will be composed of clusters 1 to n as displayed in Figures 4.6(a) and 4.6(b).

clustering structure of the Ra × Dec subspace to select 5 data aggregates to compose the positive region.

With this, we wish to verify how the performance of OptVS and previously considered AL algorithms, SM [Tong and Koller, 2001], QBD [Settles, 2012], and DSM [Huang et al., 2018], is impacted as the user interest region becomes more fragmented, that is, the number of disjoint clusters compositing the user interest increases. We also note that, as before, each algorithm will start exploration with a single randomly picked positive and negative examples, being up to each active learner to find the remaining clusters. In other words, these workloads test the different trade-offs between exploration (finding new clusters) and exploitation (accurately capturing all data examples in each cluster) of each AL strategy.

Figure 4.7 contains the experimental results for SDSS C1 and C2 with an increasing number of interest clusters. The first point we notice is that the convergence speed of all AL methods tends to decrease with the number of clusters. For example, Figures 4.7(d)–4.7(f) display the performance results for SDSS C2. In this case, while all algorithms managed to reach 100% accuracy in less than 25 iterations for a single cluster, no algorithm can attain perfect accuracy even after 200 labeled examples once 5 clusters are present. This behavior is expected since algorithms require more labeled data to find each cluster and refine its decision boundary.

Another interesting point is that OptVS and QBD tend to outperform Simple Margin and DSM by a large margin once multiple clusters are present. The poor performance of DSM is because it assumes that the user interest region is a convex set, which is no longer true. The increased performance of QBD, on the other hand, is a consequence of its selection strategy that has an exploratory component to it, which helps to find all data clusters very quickly. More interestingly, we also note that despite not explicitly having an exploratory component to its strategy, OptVS still manages to match QBD's performance and find most, if not all, interest clusters.

Summary of Experimental Results

In conclusion, we observed the following main points regarding our OptVS algorithm:

- *Performance*: OptVS was observed to outperform other VS algorithms in all Car queries and 6 out of 11 SDSS queries while still achieving comparable performance in the remaining cases. In particular, we note that OptVS can reach high accuracy with a small number of labeled points: in the case of SDSS 01–03, OptVS reached an accuracy larger than 67% after only 25 labeling iterations, while all other VS algorithms were still below 19% F-score. Besides, OptVS was also shown to outperform DSM in non-factorized scenarios.
- *Efficiency*: OptVS runs under interactive performance at all times, taking less than 1 second per iteration. In particular, our algorithm is around 10 times faster than ALuMA.
- *Multiple regions*: OptVS was also shown to outperform Simple Margin and DSM by a large margin when the user interest is composed of multiple disjoint regions, thus offering a better trade-off between exploration and exploration. For example, for SDSS C1 with 5 clusters, OptVS is at 80% accuracy after 125 iterations, while the other two are still at 45% or below.

4.5 Chapter Summary

In this chapter, we have developed an optimized implementation of our kernel version space strategy devised in Chapter 3, which results in an optimized version space algorithm called OptVS. In particular, we have introduced several optimizations for efficiently sampling the version space, which includes:

- *Sampling from a single hit-and-run chain*: Using a single hit-and-run chain for generating all samples,

- *Caching the rounding ellipsoid*: By caching the rounding ellipsoid between labeling iterations, we can warm-start the rounding computations from one iteration to the next and, consequently, drastically reduce the number of iterations until convergence.

- *A numerically stable rounding algorithm*: By exploiting the particular shape of the version space, we devised an optimized rounding algorithm that is numerically stable and avoids the expensive diagonalization step of Lovasz's method [Lovász, 1986].

Our evaluation shows that OptVS outperforms state-of-the-art version space algorithms [Tong and Koller, 2001, Settles, 2012, Gonen et al., 2013] at every iteration in 25 out of 29 user patterns considered. For example, over SDSS 01–03, our algorithm reached an accuracy higher than 67% after only 25 labeling iterations, while all other VS algorithms were still below 19% F-score. In terms of time measurements, OptVS runs under interactive performance at all times, taking less than 1 second per iteration even after hundreds of labeled points. In particular, our algorithm is around 10 times faster than ALuMA.

Chapter 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 when a user labels a data instance, the decision-making process can often be broken into a set of simple, independent "yes" or "no" questions that, when combined, 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 his questions precisely as classification models, which would require knowing the exact shape of the decision function and all constants used. In fact, we only expect the user to have a high-level intuition of the set of questions and their related attributes.

Factorization Structure. Formally, we model the user intuition of the set of questions and their 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 user intuition, Q can be broken into simple independent questions $\{Q_1, \ldots, Q_S\}$ with each Q_s posed on a subset of attributes $\boldsymbol{A}^s \subset \boldsymbol{A}$. The family of attribute sets $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^S)$ may be disjoint or overlapping in attributes as long as the user believes that decisions for these smaller questions can be made independently. $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^S)$ is referred to as the *factorization structure*.

The independence assumption in the decision process should not be confused with the data correlation issue. For example, the color and size of cars can be statistically correlated, e.g., large cars are often black. 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 his decisions for the color and size are independent, the factorization structure ({color}, {size}) applies. On the contrary, if the two attributes are not independent in the decision-making process, e.g., the user prefers red for small cars and black for large ones, but the year of production is an independent concern, then the factorization structure can be ({color size}, {year}).

In what follows, we assume the user can provide a factorization structure at the beginning of the exploration process. In the next chapter, we relax this assumption by developing an algorithm for inferring a relevant factorization from user-labeled data; see Chapter 6 for more details.

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

$$Q(x) = F(Q_1(x^1), \ldots, Q_S(x^S)) \tag{5.1}$$

The most common example of decision function is the conjunctive form, 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, $Q_1(x^1) \wedge \ldots \wedge Q_S(x^S)$ already covers a large class of decision problems and it will be considered the default choice. We note that our work can also support any other choice of decision function, but then it must be specified by the user.

Given the decision function F, we aim to learn the subspatial decision functions, $\{Q_1(x^1), \ldots, Q_S(x^S)\}$, efficiently from a small set of labeled instances. For a labeled instance x, the user provides a collection of *subspatial labels* $(y^1, \ldots, y^S) \in \{+, -\}^S$ 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 framework with similar works in the literature.

First, we note the differences in our factorization framework from Huang et al. [2018]:

- Our factorization is applied to version space algorithms, while Huang et al. [2018] does not consider them at all.
- We eliminate the assumption that either the set $\{x^s : Q_s(x^s) = +\}$ or the set $\{x^s : Q_s(x^s) = -\}$ are convex objects.
- The global decision function F must be conjunctive in Huang et al. [2018], which is relaxed to any boolean expression in our work. More precisely, our

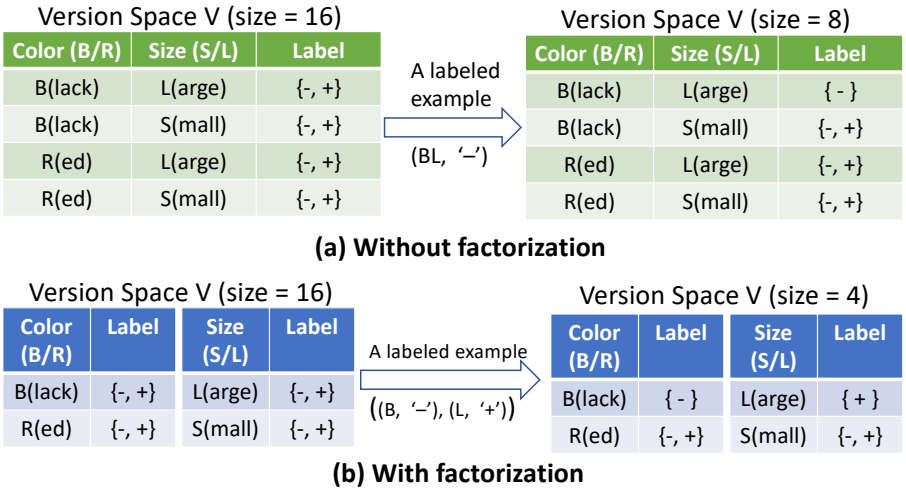


Figure 5.1: Illustration of factorization on the version space.

factorized algorithms can support any non-conjunctive decision function, but in those cases, the user should provide its particular expression at the beginning of the exploration process.

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, the user usually has to manually tune one or more parameters in such functions, which can have a large impact on the prediction quality if not properly done. 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 knowing 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, we can mention 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" remains in all of these works, including ours: in our case, the user prediction function $Q(x)$ is broken down into several simpler predictors $Q_s(x^s)$, which can then be pieced together into the final prediction $Q(x) = F(Q_1(x^1), \ldots, Q_S(x^S))$.

5.1 Introduction to Factorized Version Space

We now give an intuitive description of the factorized version space, deferring a formal definition to Section 5.3.

Without factorization, our problem is to learn a classifier h (e.g., a kernel classifier) from a labeled data set $\mathcal{L} = \{(x_i, y_i)\}$, where x_i is a data instance on the attribute set $\boldsymbol{A}$ 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 $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^S)$, we define S subspaces, with the s-th subspace including all the data instances projected on $\boldsymbol{A}^s$. Then, our goal is to learn a classifier h^s for each subspace s from its labeled set $\mathcal{L}^s = \{(x_i^s, y_i^s)\}$, y_i^s being the subspatial label for x_i^s. For the classifier h^s, its version space, $\mathcal{V}^s$, includes all possible configurations of h^s that are consistent with $\mathcal{L}^s$. Across all subspaces, we can reconstruct the version space via $\mathcal{V}_f = \mathcal{V}^1 \times \ldots \mathcal{V}^S$. For any unlabeled instance, x, we can use $F(h^1(x^1), \ldots, h^S(x^S))$ 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 5.1 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), resulting in four combinations of vehicles in total. Figure 5.1(a) shows that without factorization and in the absence of any user labeled data, the version space contains 16 possible classifiers that correspond to all possible combinations of $\{+, -\}$ 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 the 8 classifiers that assign the '$-$' label to BL.

Next, consider factorization. In the absence of labeled data, Figure 5.1(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 all combinations of $\{+, -\}$ labels over these two types of cars. Similarly, the subspace for size also includes 4 classifiers and, by combining both, we have 16 possible classifiers in total. 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 a faster reduction of the version space.

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 subsample 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: $\quad \mathcal{U}' \leftarrow \text{subsample}(\mathcal{U}, m_2)$
5: $\quad$ **for** each subspace s **do**
6: $\quad\quad \mathcal{U}^s \leftarrow \{x^s, \text{ for } x \in \mathcal{U}'\}$
7: $\quad\quad \mathcal{L}^s \leftarrow \{(x^s, y^s), \text{ for } (x, y) \in \mathcal{L}\}$
8: $\quad\quad \{p^s_{x,+}(x)\}_{x \in \mathcal{U}'} \leftarrow \text{compute\_cut\_probability}(\mathcal{L}^s, \mathcal{U}^s, M)$
9: $\quad$ **end for**
10: $\quad x^* \leftarrow \text{selection\_strategy}(\mathcal{U}', p^1_{x,+}, \ldots, p^S_{x,+})$
11: $\quad y^* \leftarrow \text{get\_labels\_from\_user}(x^*)$
12: $\quad \mathcal{L}, \mathcal{U} \leftarrow \mathcal{L} \cup \{(x^*, y^*)\}, \mathcal{U}/\{x^*\}$
13: **end while**
14: **for** k from 1 to S **do**
15: $\quad MV^s \leftarrow \text{train\_majority\_vote\_classifier}(\mathcal{L}^s)$
16: **end for**
17: $h_{final} \leftarrow x \mapsto F(MV^1(x^1), \ldots, MV^S(x^S))$
18: **return** h_{final}

5.2 Overview of a Factorized Version Space Algorithm

Based on the above insight, we propose a new active learning algorithm called the *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 each subspace.

Algorithm 4 shows the pseudocode 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 to guarantee the interactive performance of our algorithm, as discussed in Section 3.5. Then it proceeds to an iterative procedure (lines 3-13): In each iteration, the unlabeled pool may be further subsampled to expedite learning, thus obtaining $\mathcal{U}'$ (line 4). Then the algorithm considers each subspace (lines 5-9), including both labeled instances, $(x^s, y^s) \in \mathcal{L}^s$, and unlabeled instances, $x^s \in \mathcal{U}^s$, projected to this subspace. The key step is to compute, for each unlabeled instance x, how much its projection x^s can reduce the version space $\mathcal{V}^s$ once its label is acquired (line 8). This step requires efficient sampling of the version space $\mathcal{V}^s$, as shown in Section 4. Once the above computation completes for all subspaces, then the algorithm chooses the next unlabeled instance that can result in the best reduction of the factorized version

space $\mathcal{V}_f = \mathcal{V}^1 \times \ldots \times \mathcal{V}^S$ (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 MV^s for each subspace, the algorithm builds the final classifier $F(MV^1, \ldots, MV^S)$ (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 $(\boldsymbol{A}^1, \cdots, \boldsymbol{A}^S)$ is given. For each subspace s, the user labels the projection x^s of x over $\boldsymbol{A}^s$ based on a hypothesis from a hypothesis class $\mathcal{H}^s$ with prior probability distribution π^s. The user then provides a binary label $\{+, -\}$ for each subspace; in other words, for each x a collection of *subspatial labels* $(y^1, \ldots, y^S) \in \mathcal{Y}_f = \{+, -\}^S$ is provided by the user.

Definition 5.1 (Factorized hypothesis and factorized hypothesis space). *Define a factorized hypothesis as any function $H : \mathcal{X} \to \mathcal{Y}_f$ such that*

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

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

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

Definition 5.2 (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 $\prod_s \mathcal{V}^s = \mathcal{V}^1 \times \ldots \times \mathcal{V}^S$, where

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

is the version space at subspace s.

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

Definition 5.3 (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*

$$\arg\max_{x \in \mathcal{U}} 1 - \sum_{y \in \mathcal{Y}_f} p_{x,y}^2 \tag{5.2}$$

Additionally, by noting that $\min_{H \in \tilde{\mathcal{H}}} \tilde{\pi}(H) = \prod_s \min_{h^s \in \mathcal{H}^s} \pi^s(h^s)$, the following theorem is a direct application of Equation (3.2):

Theorem 5.4 (Number of iterations with factorization). *The factorized greedy strategy in* (5.2) *takes at most*

$$OPT_f \cdot \left(1 + \sum_{s=1}^{S} \log\left(\frac{1}{\min_{h^s} \pi^s([h^s])}\right)\right)^2$$

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.*

It is interesting to notice that OPT_f requires the classifiers to continue exploring until *the user's concept is found in every subspace*. As one can intuitively expect, the value of OPT_f is governed by the most complex subspace in the factorization in terms of data distribution and complexity of the hypothesis class $\mathcal{H}^s$.

Finally, we derive a simplified computation of the factorized greedy strategy (5.2):

Theorem 5.5. *Let* $p^s_{x,\pm} = \mathbb{P}(h(x^s) = \pm \,|\, [h] \in \mathcal{V}^s)$, *the factorized greedy selection strategy (5.2) is equivalent to*

$$\arg\max_{x \in \mathcal{U}} 1 - \prod_{s=1}^{S} (1 - 2\, p^s_{x,+} p^s_{x,-}) \tag{5.3}$$

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

$$\begin{aligned} p_{x,y} &= \mathbb{P}(H(x) = y \,|\, [H] \in \mathcal{V}_f) \\ &= \mathbb{P}(h^s(x^s) = y^s,\, \forall s \,|\, [h^s] \in \mathcal{V}^s,\, \forall s) \\ &= \prod_s \mathbb{P}(h^s(x^s) = y^s \,|\, [h^s] \in \mathcal{V}^s) \\ &= \prod_s p^s_{x,y^s} \end{aligned}$$

Therefore, we obtain

$$\begin{aligned}\sum_{y\in\mathcal{Y}_F} p_{x,y}^2 &= \sum_{y_1=\pm}\cdots\sum_{y_S=\pm}\prod_s (p_{x,y^s}^s)^2 \\ &= \prod_s \sum_{y^s=\pm} (p_{x,y^s}^s)^2 \\ &= \prod_s \left((p_{x,+}^s)^2 + (p_{x,-}^s)^2\right) \\ &= \prod_s (1 - 2p_{x,+}^s p_{x,-}^s)\end{aligned}$$

□

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:

$$\arg\max_{x\in\mathcal{U}} \sum_{s=1}^{S} 2p_{x,+}^s p_{x,-}^s \tag{5.4}$$

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

Theorem 5.6 (Factorized squared-loss strategy). *The squared-loss strategy takes at most*

$$S \cdot OPT_f \cdot \left(1 + \sum_{s=1}^{S} \log\left(\frac{1}{\min_{h_s} \pi_s([h_s])}\right)\right)^2$$

iterations in expectation.

Proof. For $p = (p_1, \cdots, p_S)$, let's define $f(p) = 1 - \prod_s (1 - 2p_s(1-p_s))$ and $g(p) = \sum_s 2p_s(1-p_s)$. Additionally, set $x_f^* = \arg\max_x f(p(x))$ and $x_g^* = \arg\max_x g(p(x))$, where $p(x) = (p_{x^1,+}, \cdots, p_{x^S,+})$. We wish to show that $f(x_g^*) \geq \frac{1}{S} f(x_f^*)$, from which our the main result follows from Theorem 13 of Golovin and Krause [2011].

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

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

Now, set $q_s = 1 - 2p_s(1-p_s) \in [\frac{1}{2}, 1]$, from which we obtain $f(q) = 1 - \prod_s q_s$ and $g(q) = S - \sum_s q_s$. By the arithmetic-geometric inequality we have

$$\prod_s q_s \leq \left(\prod_s q_s \right)^{1/S} \leq \frac{1}{S} \sum_s q_s,$$

which gives $f(q) \geq g(q)/S$.

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

$$h(t) = t \prod_{s=1}^{S} q_s - t - \sum_{s=1}^{S} q_s + S$$

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

5.5 Factorized Product-Loss Strategy

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

$$\arg\max_{x \in \mathcal{U}} 2\tilde{p}_{x,+}\tilde{p}_{x,-} \tag{5.5}$$

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

$$\tilde{p}_{x,+} = \prod_{s=1}^{S} p^s_{x,+}$$

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 subspatial 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.

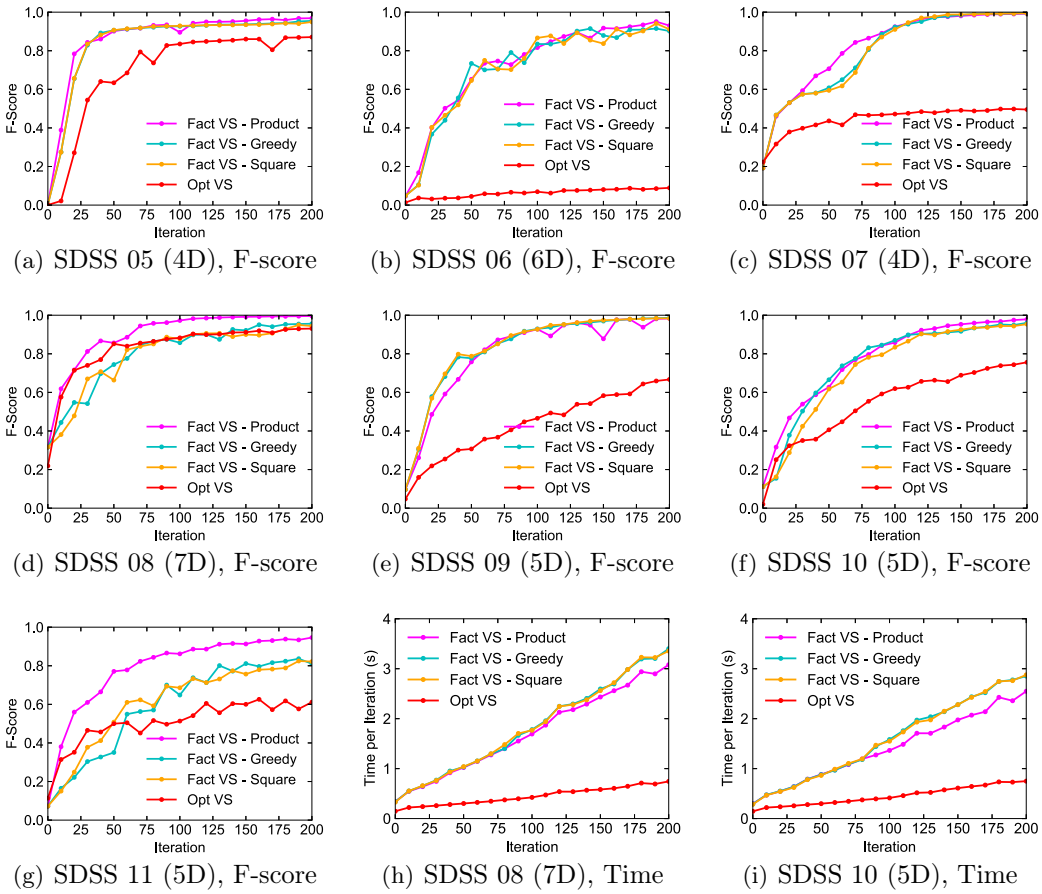
5.6 Optimization for Categorical Subspaces

The factorization framework also leads to a simple treatment of categorical variables. Assume that all attributes are categorical in a given subspace s, and let $\mathcal{C} = \{x^s : x \in \mathcal{X}\}$ be the set of all different categories in the data. Instead of encoding these values to numbers (e.g., using the one-hot encoding), we define a categorical hypothesis class $\mathcal{H}^{cat} = \{h : \mathcal{C} \to \{+,-\}\}$ to model the user interest over this subspace; in other words, each classifier represents a $\{+,-\}$-vector over categories indicating which are interesting or not. In particular, it is easy to implement the version space bisection rule over $\mathcal{H}^{cat}$: given some labeled data $\mathcal{L}^s$ and by assuming a uniform prior, the version space cut probabilities are easily shown to be

$$p_{c,+} = \begin{cases} 1, \text{ if } (c,+) \in \mathcal{L}^s \\ 0, \text{ if } (c,-) \in \mathcal{L}^s \\ 0.5, \text{ otherwise} \end{cases} \tag{5.6}$$

which can be directly plugged into Equations (5.3) or (5.4). As for prediction, we simply use the majority vote classifier definition $MV^{cat}(c) = + \iff p_{c,+} > 0.5 \iff (c,+) \in \mathcal{L}$.

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.



(a) SDSS 05 (4D), F-score (b) SDSS 06 (6D), F-score (c) SDSS 07 (4D), F-score

(d) SDSS 08 (7D), F-score (e) SDSS 09 (5D), F-score (f) SDSS 10 (5D), F-score

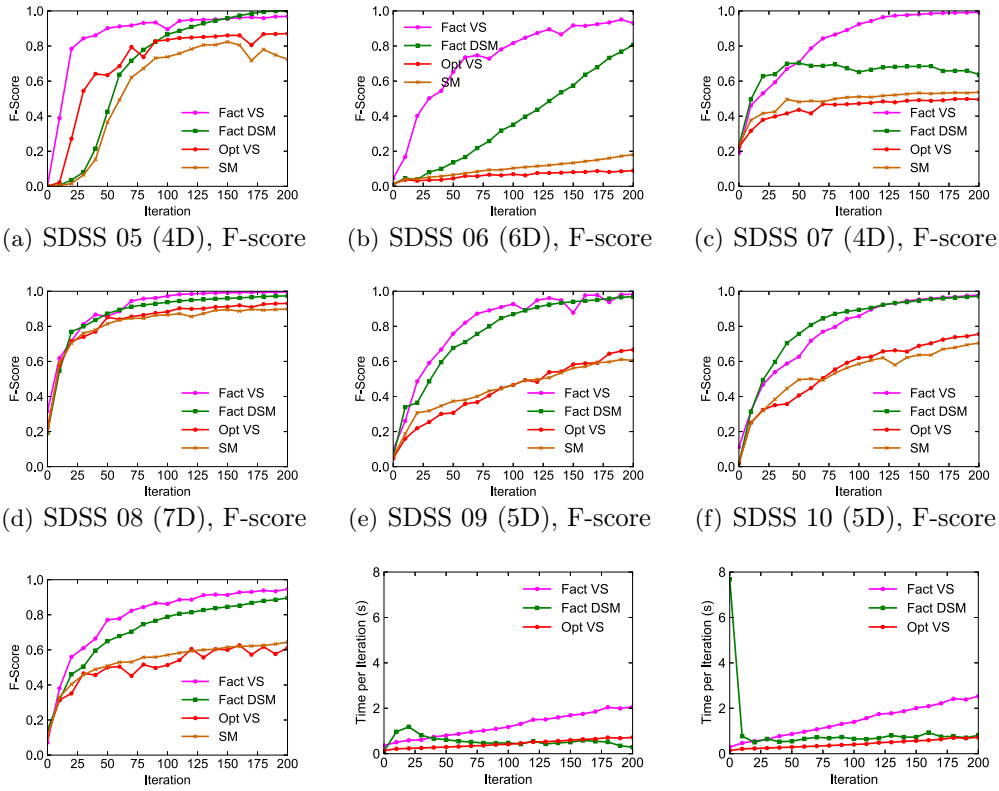
(g) SDSS 11 (5D), F-score (h) SDSS 08 (7D), Time (i) SDSS 10 (5D), Time

Figure 5.2: Effects of factorization on performance using the SDSS data set. We also display in parenthesis the dimensionality of each query.

5.7 Experimental Results

To end this chapter, we showcase how our factorized version space algorithm, denoted by "Fact VS", fares in practice against non-factorized methods like OptVS and Simple Margin Tong and Koller [2001], as well as Factorized DSM [Huang et al., 2018], a factorization-aware active learning algorithm. We do not consider QBD [Settles, 2012] and ALuMA [Gonen et al., 2013] since OptVS outperforms them in almost every case.

We evaluate our techniques under the same experimental conditions described in Section 4.4, with only a few differences. First, we do not compare over the SDSS 01–04 because these queries are composed of a single subspace, causing Fact VS to degenerate into OptVS. Second, all factorization-aware algorithms assume a conjunctive labeling function for prediction. Finally, our factorized VS



(a) SDSS 05 (4D), F-score (b) SDSS 06 (6D), F-score (c) SDSS 07 (4D), F-score

(d) SDSS 08 (7D), F-score (e) SDSS 09 (5D), F-score (f) SDSS 10 (5D), F-score

(g) SDSS 11 (5D), F-score (h) SDSS 06 (6D), Time (i) SDSS 11 (5D), Time

Figure 5.3: Comparison of Fact VS with other active learning techniques [Tong and Koller, 2001, Huang et al., 2018] in terms of classification accuracy and time per iteration using the SDSS data set. We also display in parenthesis the dimensionality of each query.

algorithm uses the same hyper-parameters as OptVS for sampling each version subspace, except for a slightly smaller sample size of 16 and 100 burn-in steps.

Evaluating our Techniques Using SDSS

Expt 1 (Factorization): We first study the effect of factorization by comparing OptVS with its extension to factorization, Fact VS. For factorization, each predicate in the target query corresponds to its own factorized subspace. We also note that 05–07 present no overlap of attributes across subspaces, while 08–11 present such an overlap.

For SDSS 05–11, Fact VS outperforms OptVS by a wide margin as displayed in Figures 5.2(a)-5.2(g). In all cases, factorization can provide a significant boost in performance; for 06 in particular, the factorized version reaches $> 90\%$ F-score

Query	Fact VS			Fact DSM			OptVS			SM		
	25	50	100	25	50	100	25	50	100	25	50	100
05	**82**	**90**	**90**	6	43	87	38	63	84	3	37	74
06	**42**	**65**	**82**	6	14	35	3	4	7	5	7	10
07	**56**	**71**	**92**	62	70	65	40	44	47	44	48	51
08	73	86	**97**	**76**	**87**	94	80	85	88	70	81	86
09	**54**	**76**	**93**	43	68	87	21	31	47	32	37	47
10	50	63	86	**55**	**76**	**89**	32	41	62	34	50	59
11	**62**	**77**	**86**	50	65	79	43	50	51	43	51	57

Table 5.1: F-score (in %) comparison at iterations 25, 50, and 100 of Fact VS with other active learning techniques [Huang et al., 2018, Tong and Koller, 2001] using the SDSS data set. Bold values represent the observed maximum across all algorithms per query at the given iteration.

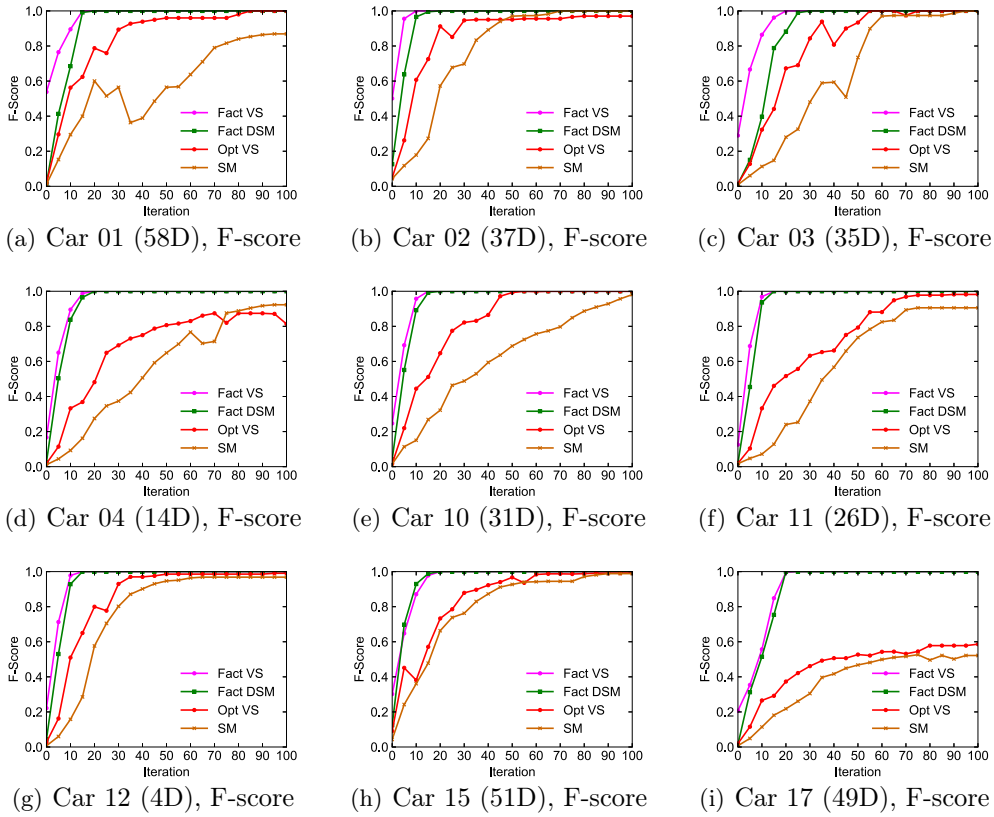
after 200 iterations while the non-factorized version is still at $< 10\%$. Also, we do not observe any performance difference between the greedy and squared-loss strategies of factorization, despite the latter having a worse theoretical bound. As for the product loss, results are nearly identical to the other two losses for most queries, except 08 and 11 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.

Finally, the running time of Fact VS is somewhat higher than OptVS due to the handling of multiple subspaces and our current implementation that runs subspatial computations serially. Figures 5.2(h) and 5.2(i) show the time for SDSS 08 and 10, which are the two most expensive queries with 5 and 6 subspaces. 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.

Comparing with Other Techniques Using SDSS

Expt 2 (Comparison to other algorithms): We next compare our factorized algorithm to three active learning strategies: OptVS, Simple Margin [Tong and Koller, 2001], and a factorization-aware algorithm, Factorized DSM [Huang et al., 2018]. Our Factorized Version Space algorithm uses the product loss, which tends to give better results. Figure 5.3 illustrates per-iteration measurements for all high-dimensional queries, while Table 5.1 shows a resumed set of results.

Our Fact VS almost always outperforms others, including DSM that uses factorization and stronger assumptions. Figures 5.3(b) and 5.3(g) show the results for 06 and 11. In general, Fact VS outperforms DSM, which in turn is an upper



(a) Car 01 (58D), F-score (b) Car 02 (37D), F-score (c) Car 03 (35D), F-score

(d) Car 04 (14D), F-score (e) Car 10 (31D), F-score (f) Car 11 (26D), F-score

(g) Car 12 (4D), F-score (h) Car 15 (51D), F-score (i) Car 17 (49D), F-score

Figure 5.4: Comparison of Fact VS with other active learning techniques [Tong and Koller, 2001, Huang et al., 2018] in terms of classification accuracy using the Car data set. We also display in parenthesis the dimensionality of each query after one-hot encoding.

bound of all other non-factorized algorithms. In the particular case of 5.3(b), 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 5.3(h) and 5.3(i) show the running time of Fact VS, DSM, and OptVS for SDSS 06 and 11. 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. Moreover, 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 the initial iterations.

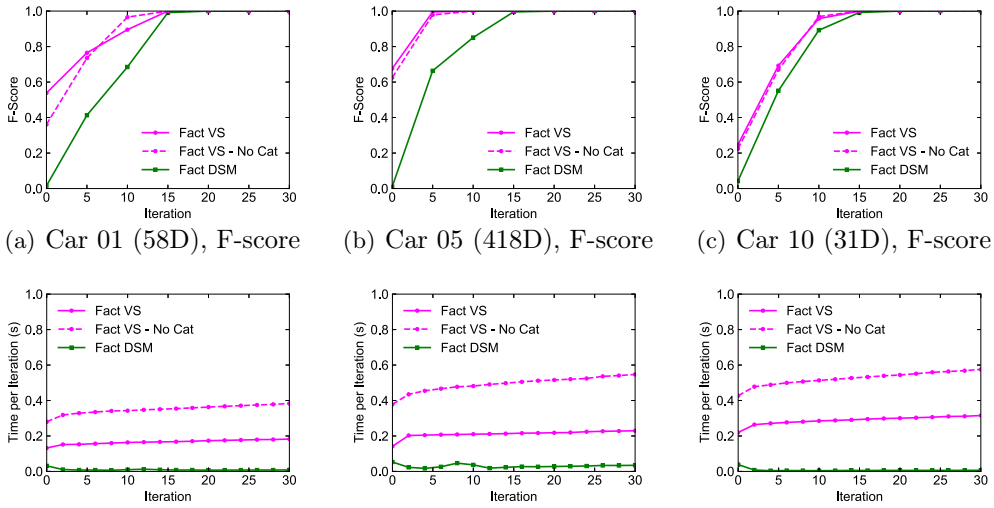
Query	Fact VS			Fact DSM			OptVS			SM		
	5	10	20	5	10	20	5	10	20	5	10	20
01	**76**	**90**	**100**	41	69	**100**	30	56	79	15	29	60
02	**96**	**100**	**100**	64	97	**100**	26	61	91	12	18	57
03	**67**	**86**	**100**	15	40	88	13	32	67	6	11	28
04	**65**	**90**	**100**	50	84	**100**	11	33	48	4	9	27
05	**100**	**100**	**100**	66	85	**100**	57	89	**100**	31	43	66
06	**84**	**100**	**100**	73	**100**	**100**	52	90	**100**	39	73	**100**
07	**75**	89	**100**	56	**90**	**100**	45	72	77	18	38	56
08	**100**	**100**	**100**	**100**	**100**	**100**	95	**100**	**100**	**100**	**100**	**100**
09	89	**100**	**100**	**92**	**100**	**100**	61	76	98	51	71	99
10	**69**	**96**	**100**	55	89	**100**	22	44	65	11	15	32
11	**69**	**97**	**100**	45	94	**100**	10	33	52	5	7	24
12	**71**	**98**	**100**	53	93	**100**	16	51	80	6	16	58
13	**90**	99	**100**	83	**100**	**100**	22	46	78	18	31	51
14	**94**	**100**	**100**	84	**100**	**100**	33	51	94	30	41	65
15	65	87	**100**	**70**	**93**	**100**	45	38	73	24	36	66
16	**99**	**100**	**100**	94	**100**	**100**	50	86	**100**	35	53	78
17	**35**	**56**	**100**	31	51	**100**	12	27	37	5	11	22
18	**98**	**100**	**100**	97	**100**	**100**	64	94	91	57	82	96

Table 5.2: F-score (in %) comparison at iterations 5, 10, and 20 of Fact VS with other active learning techniques [Huang et al., 2018, Tong and Koller, 2001] using the Car data set. Bold values represent the observed maximum across all algorithms per query at the given iteration.

Comparing with Other Techniques Using Car Queries

For the 18 queries over the Car 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 the data sparsity, which allows more freedom in separating the positive and negative classes. Refer to Figure 5.4 and Table 5.2 for a complete set of results.

Expt 3 (Optimization for categorical attributes): We next study the effect of our optimization for categorical variables, as described in Section 5.6. Figure 5.5 shows the result for the three Car queries with the highest number of categorical subspaces. 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 at least 100% for all these queries. We also observed similar results over all other car queries.



(a) Car 01 (58D), F-score (b) Car 05 (418D), F-score (c) Car 10 (31D), F-score

(d) Car 01 (58D), Time (e) Car 05 (418D), Time (f) Car 10 (31D), Time

Figure 5.5: Evaluating our optimization for categorical subspaces using the Car data set. We also display in parenthesis the dimensionality of each query after one-hot encoding.

Summary of Experimental Results

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

- *Advantages of factorization*: Fact VS was shown to outperform OptVS in almost all high-dimensional cases, irrespectively of the strategy being used. Furthermore, we did not observe any performance difference between the greedy and the squared-loss strategies, but we did observe a few cases in which the product loss significantly outperforms the other two.

- *Comparing with other techniques*: For higher dimensional problems, our Fact VS algorithm significantly outperforms other VS algorithms as well as DSM, a factorization-aware algorithm, while maintaining interactive speed. For example, for a complex user interest pattern tested, our algorithm achieves an 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 Fact VS and DSM is reduced for the simple Car data set due to the data sparsity and, consequently, fast convergence.

- *Efficiency*: In terms of time measurements, Fact VS tends to be faster than DSM in initial labeling iterations while still running under 4 seconds per iteration at all times.

- *Optimization for categorical subspaces*: Our optimization for categorical subspaces was shown to significantly improve the running time of our algorithms without compromising the accuracy, leading to improvements of up to 100% in running time.

5.8 Chapter Summary

In this chapter, we have introduced the concept of a factorization structure, a decomposition of the user decision-making process into sub-questions whose answers can be combined to derive the final label. With this new piece of information, we have devised the following main contributions:

- *A Factorized Version Space algorithm*: We designed a novel *Factorized Version Space* algorithm (Fact VS) capable of decomposing the version space into several low-dimensional subspaces over which exploration is much more efficient. Compared to previous work on factorization [Huang et al., 2018], our method lifts the restrictions of conjunctive decision function as well as convexity of the user interest pattern in each subspace.
- *Factorized exploration strategies*: Our factorized algorithm also supports three different selection strategies: Greedy, Squared-Loss, and Product-Loss strategies. In particular, the first two losses come with strong theoretical guarantees on performance, while the last one offers a better convergence speed in practice.
- *Optimization for categorical subspaces*: Additionally, we have also developed a new optimization for categorical subspaces that replaces the VS sampling scheme with efficient memorization of categories and their labels, leading to improvements of up to 100% in running time.
- *Evaluation*: Our evaluation results show that, for all 25 user patterns considered, Fact VS outperforms non-factorized active learners, including OptVS, as well as a factorization-aware algorithm, DSM [Huang et al., 2018], often by a wide margin while still running under interactive performance. For example, for a complex user interest pattern tested, our algorithm achieves an 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.

Chapter 6

Learning a Factorized Classifier

In the previous chapter, we have seen that factorized active learners can significantly outperform traditional active learning strategies. Once a factorization structure is available, factorized algorithms are able to decompose the high-dimensional learning problem as a collection of low-dimensional learning tasks, which are much more efficient to solve. However, by taking a closer look at the motivations and assumptions behind factorization, a few relevant questions become apparent:

What if the user does not know the factorization? To provide a factorization structure, the user needs a deep understanding of his own decision-making process and familiarity with the data distribution, which may not be possible before the user has reviewed a fairly large number of data examples. In these cases, the user may prefer to leave to the system to automatically infer a factorization structure from a modest set of labeled data.

Can we learn a classifier that mimics the user decision-making process? The user studies of our previous works [Huang et al., 2018, 2019] show that when the human user is asked to retrieve data points of interest from large data sets, the user often breaks his decision into smaller, independent questions whose answers can be combined to make the final answer. In particular, this observation suggests the design of a novel *factorized classification model*, capable of breaking its prediction process as a collection of low-dimensional, independent classification tasks. If successfully trained, such classifiers can potentially bring several benefits:

- *Accuracy*: In the same way that factorized active learners brought a huge performance boost compared to non-factorized approaches, we expect that a factorized classifier will also bring similar performance advantages in human-centered learning tasks.
- *Efficiency*: Training can also be made efficient as the original complex learning problem is broken into a few simpler ones.
- *Interpretability*: By design, the factorized classifier mimics the human

decision-making process, making its predictive process easy to interpret by practitioners. More precisely, the model's interpretability is a consequence of three main factors: (1) the low-dimensionality of individual subspaces, (2) the use of original features in each subspace which are semantically meaningful, and (3) an intuitive logic for combining all independent decisions into the final label prediction.

In conclusion, this chapter aims to *design a learning algorithm for factorized classifiers* that infers both a simple factorization structure and the local classifiers in each subspace from the labeled data. We organize the discussion as follows: Section 6.1 considers the problem of learning a factorized classifier in the *batch setting*, where a large pool of labeled data is available. After that, Section 6.2 applies this knowledge to the *active learning scenario*, devising an automatically factorized exploration strategy that does not require any extra information from the user. Finally, Sections 6.3 and 6.4 close the chapter with a collection of experimental results, comparing our approach to state-of-the-art algorithms in both the batch and active learning cases.

6.1 Learning a Factorized Classifier

In this section, we develop a human-inspired classification model capable of breaking the labeling process into a collection of simple predictions, in the same way that users can simplify their own decision-making through a factorization structure. In particular, we propose a novel factorized classifier, called the Factorized Linear Model (FLM), that can decompose the user interest pattern as a combination of convex objects in low-dimensional subspaces. We also present a few extensions of this method that enhance its applicability, including an extension to kernel classifiers and optimizations for categorical variables. To simplify the design of our new techniques, we begin by considering the *batch learning case*, where a large pool of labeled points is available for training. We consider this simplified case not only as a means to start the algorithm design but also because of the practical benefits a factorized algorithm can bring in this setting, as we will show in the performance study in Section 6.3. We defer the extension to the active learning setting to Section 6.2.

Intuitively speaking, a factorized classifier should be able to break the labeling process as a collection of several independent classifications, each one depending on a small selection of attributes. This intuition is more precisely captured by the following definition:

Definition 6.1 (Factorized probabilistic model). *Let* $(X, Y) \subset \mathbb{R}^d \times \{0, 1\}$ *be the random variable modeling the labeled data distribution. A factorized probabilistic model for the label* Y *is a collection of four elements:*

1. *A number of subspaces* $S \in \mathbb{N}$
2. *A decision function* $F : \{0,1\}^S \to \{0,1\}$
3. *A factorization structure* $(\boldsymbol{A}^1, \ldots, \boldsymbol{A}^S)$*, with* $\boldsymbol{A}^s \subset \boldsymbol{A} = \{1, \ldots, d\}$
4. *A set of subspace labels* $\{Y^1, \ldots, Y^S\}$ *satisfying:*
   - $\{Y^1, \ldots, Y^S\}$ *are independent random variables*
   - $Y = F(Y^1, \ldots, Y^S)$
   - *Each* Y^s *is independent of* X_i *for* $i \notin \boldsymbol{A}^s$*, that is,* $\mathbb{P}(Y^s = y \,|\, X = x) = \mathbb{P}(Y^s = y \,|\, X^s = x^s)$ *where* $x^s = proj(x, \boldsymbol{A}^s)$

In other words, the above probabilistic model decomposes the final user label Y as a function of S independent decisions $Y = F(Y^1, \ldots, Y^S)$, where each Y^s depends only on a subset of features $\boldsymbol{A}^s$. However, compared to the previous chapter, training such a factorized classifier is a much more complex problem due to a couple reasons:

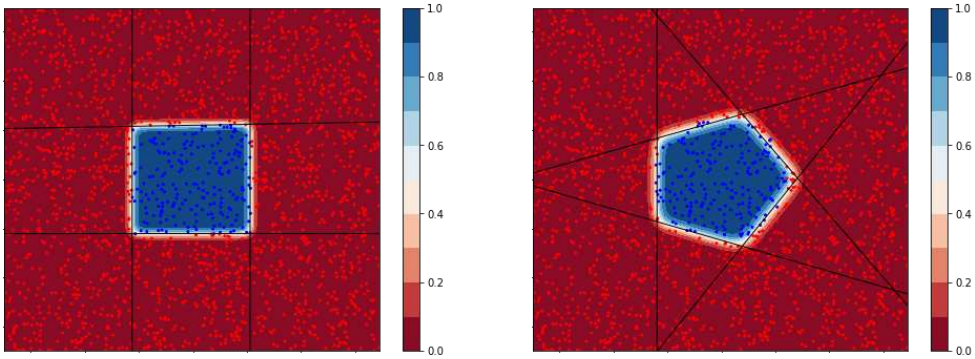
- *Subspace labels are unknown*: Factorized active learners assume to possess the subspace labels Y^s for each data point, which effectively allows them to decompose the learning problem into several simple classifications. However, the factorized classifier can only utilizes the final user label Y for training, with the subspace labels having to be inferred by the learning algorithm.
- *Factorization is unknown*: Another assumption of factorized active learners is that the factorization structure is known in advance. On the other hand, factorized classifiers must learn a factorization from the labeled data, which requires estimating both the number of subspaces S and the most relevant features per subspace $\boldsymbol{A}^s$.

In essence, a factorized model must estimate several interdependent components from the labeled data, which considerably increases the complexity of the learning problem. To make it more tractable, we put a couple of restrictions in place, starting with the following:

Axiom 6.2 (Conjunctive assumption). *We assume a conjunctive decision function* F*, that is:*

$$F(y^1, \ldots, y^S) = y^1 \wedge \ldots \wedge y^S = \min_s y^s \tag{6.1}$$

Intuitively, the conjunctive assumption attempts to model the user decision process as a list of requirements: a data point is labeled as "interesting" if and only if it meets all necessary constraints. However, this assumption is not as



(a) Square pattern: $max(|x_1|, |x_2|) < L$ (b) Circle pattern: $x_1^2 + x_2^2 < R^2$

Figure 6.1: Examples of factorized linear models accurately learning non-linear interest patterns. Blue and red dots represent positive ($Y = 1$) and negative ($Y = 0$) data points, and contour lines the predictive positive class probability. Black lines correspond to the decision boundaries of each subspatial linear model: $\{x : b^s + \langle x, w^s \rangle = 0\}$.

restrictive as it looks. First, every boolean function F can be written in the Conjunctive Normal Form (CNF), meaning we can always fall back to the conjunctive case by increasing the complexity of Y^s and $\boldsymbol{A}^s$. Second, this assumption is backed by the results of our recent user studies [Huang et al., 2018, 2019] in which most of the observed user exploration queries are conjunctions of simple predicates; see Appendix D for an overview of all queries. Finally, previous works on factorization [Huang et al., 2018] also relied on the conjunctive assumption.

Now, we consider the problem of how to model the subspace label Y^s. Assuming that the user decision-making process in each subspace is very simple, we will attempt to model it via a simple classifier as well, the linear one:

Axiom 6.3 (Linearity assumption). *The subspace labels Y^s can be accurately modeled by logistic regression classifiers; more precisely, for each subspace s there is $(b^s, w^s) \in \mathbb{R} \times \mathbb{R}^d$ satisfying:*

$$\mathbb{P}(Y^s = 1 | X = x) = \sigma(b^s + \langle x, w^s \rangle) \tag{6.2}$$

where $\sigma(u) = 1/(1 + e^{-u})$ is the sigmoid function.

Choosing to model the subspace labels as linear classifiers offers several advantages:

- *Support of complex user interest patterns*: Despite sounding restrictive, the linearity assumption still allows for complex interest patterns to be learned. More precisely, by combining the predictions of multiple linear models, it

can construct a polytope-like decision boundary that can even approximate non-linear shapes; see Figure 6.1 for an illustration of this behavior. For a more complete discussion on this topic, please refer to Section 6.1.3.

- *Easy estimation of the factorization structure*: It is simple to see that a subspace label Y^s is independent of X_i whenever $w_i^s = 0$. Consequently, the factorization structure can be easily extracted by looking at the sparsity pattern of the weight vectors:

$$\boldsymbol{A}^s = \{i \in \boldsymbol{A} : w_i^s \neq 0\} \tag{6.3}$$

  In particular, irrelevant subspaces can also be identified as zero weight vectors $w^s = 0$, which helps us infer the ideal number of subspaces S.

- *Efficient optimization*: Being a parameterized model, it enables us to use highly efficient optimization algorithms for training.

Finally, by putting all the above considerations together we can now define our factorized model of choice:

Definition 6.4 (Factorized Linear Model (FLM)). *Let $S \in \mathbb{N}$. Also, consider a weight matrix $W \in \mathbb{R}^{S \times d}$ and a bias vector $b \in \mathbb{R}^S$. Then, a factorized linear model is a probabilistic classifier for the binary label Y of the form:*

$$\mathbb{P}(Y = 1 | X = x) = p_{b,W}(x) = \prod_{s=1}^{S} \sigma(b^s + \langle x, w^s \rangle) \tag{6.4}$$

where w^s is the s-th row of W.

In summary, the Factorized Linear Model represents a factorized probabilistic model of the user labeling in which subspace labels are modeled after logistic regression classifiers. Furthermore, this model naturally encodes a factorization structure via the sparsity pattern in the weight matrix W: zero entries represent irrelevant features, while full-zero rows encode irrelevant subspaces. One final point to note is that the FLM is a *generalization* of the well-known Logistic Regression model, which can be trivially recovered by setting $S = 1$.

In what follows, we study how to train an accurate FLM, including how to enforce sparsity constraints and retrieve a meaningful factorization structure.

6.1.1 Training an Accurate Factorized Model

We now turn to the problem of training an accurate FLM classifier from a pool of labeled data, leaving the sparsity constraints to the next section. In particular, we note that an FLM can only infer a meaningful factorization if it also accurately captures the labeled data distribution, that is, achieves a low generalization error.

Let $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m$ be a (possibly large) set of labeled points. Since the FLM is a probabilistic classifier, we can obtain an accurate model by simply minimizing the *cross-entropy loss*:

Definition 6.5 (Cross-entropy loss for FLM). *Let $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{0, 1\}$ be a collection of labeled data points. The cross-entropy loss function for the Factorized Linear Model is defined as:*

$$\ell(b, W) = \frac{1}{m} \sum_{i=1}^{m} \ell_{x_i, y_i}(b, W) \tag{6.5}$$

where $\ell_{x,y}$ is defined as:

$$\ell_{x,y}(b, W) = -y \log p_{b,W}(x) - (1 - y) \log(1 - p_{b,W}(x)) \tag{6.6}$$

Cross-entropy-based loss functions have been extensively studied in the Machine Learning and Statistics domains, especially in the context of training probabilistic classifiers [Murphy, 2012]. In particular, this loss has been successfully employed for training Logistic Regression and Neural Network models.

With an objective function defined, all that remains is to compute the optimal parameters minimizing the loss function ℓ. In this regard, we first prove a few relevant properties of the loss function:

Theorem 6.6. *The cross-entropy loss functions ℓ and $\ell_{x,y}$ from Equations (6.5) and (6.6) satisfy the following properties:*

1. *$\ell_{x,y}$ and ℓ are smooth (infinitely-differentiable) functions over $\mathbb{R}^S \times \mathbb{R}^{S \times d}$.*
2. *The positive probability function $p_{b,W}$ has its value unchanged by arbitrarily permuting the order of subspaces. In particular, the same is true for both $\ell_{x,y}$ and ℓ.*
3. *$\ell_{x,y}$ is a convex function if, and only if, the $S \times S$ matrix:*

   $$A_{rs} = (1 - y) p_{b,W}(x) q^r_{b,W}(x) q^s_{b,W}(x) - \delta_{rs} (p_{b,W}(x) - y) p^r_{b,W}(x) q^r_{b,W}(x)$$

   is semi-positive definite for every (b, W), where $p^s_{b,W}(x) = \sigma(b^s + \langle x, w^s \rangle)$, $q^s_{b,W}(x) = \frac{1 - p^s_{b,W}(x)}{1 - p_{b,W}(x)}$, and $\delta_{rs} = \mathbb{1}(r = s)$.
4. *For $S \geq 2$ and any $x \in \mathbb{R}^d$, $\ell_{x,y}$ is a convex function if $y = 1$ and a non-convex function if $y = 0$.*

Proof. We prove each item separately:

Item 1: It is enough to show that $\ell_{x,y}$ is a smooth function since a linear combination of smooth functions is also smooth. First, we observe that $\ell_{x,y}$ is

well-defined for all values of (b, W) since $p_{b,W} \in (0, 1)$. As for the differentiability condition, it is easy to see that $\ell_{x,y}$ is a composition of smooth functions: logarithmic, sigmoid, and linear, being, therefore, smooth as well.

Item 2: This is a simple consequence of the fact that $p_{b,W}(x) = \prod_{s=1}^{S} \sigma(b^s + \langle x, w^s \rangle)$ remains unchanged whenever any two (b^s, w^s) and $(b^{s'}, w^{s'})$ have their places swapped.

Item 3: In this proof, we simplify the notation by introducing two new quantities: $\tilde{W}_i = (b_i, W_i)$ and $\tilde{x} = (1, x)$.

Since $\ell_{x,y}$ is a twice-differentiable function, it is convex if, and only if, its Hessian matrix $Hess(\ell_{x,y})$ is positive semi-definite over all points in the domain [Boyd and Vandenberghe, 2004, Section 3.1.4]. In particular, this condition is equivalent to:

$$\min_{U \in \mathbb{R}^{S \times (d+1)}} \sum_{ri,sj} U_{ri} U_{sj} \frac{\partial^2 \ell_{x,y}}{\partial \tilde{W}_{ri} \partial \tilde{W}_{sj}}(\tilde{W}) \geq 0, \text{ for a } \tilde{W}$$

However, through elementary computations, one can show that:

$$\frac{\partial^2 \ell_{x,y}}{\partial \tilde{W}_{ri} \partial \tilde{W}_{sj}}(\tilde{W}) = A_{rs} \tilde{x}_i \tilde{x}_j$$

with A_{rs} as defined in the Theorem. In particular, by noticing that $\tilde{x} \neq 0$ implies $\{U\tilde{x} : U \in \mathbb{R}^{S \times (d+1)}\} = \mathbb{R}^S$, we can conclude that:

$$\begin{aligned} \min_{U \in \mathbb{R}^{S \times (d+1)}} \sum_{ri,sj} U_{ri} U_{sj} \frac{\partial^2 \ell_{x,y}}{\partial \tilde{W}_{ri} \partial \tilde{W}_{sj}}(\tilde{W}) &= \min_{U \in \mathbb{R}^{S \times (d+1)}} \sum_{rs} A_{rs} (U\tilde{x})_r (U\tilde{x})_s \\ &= \min_{v \in \mathbb{R}^S} v^T A v \end{aligned}$$

which finishes the proof.

Item 4: From the previous result, we simply have to check the sign of $v^T A v$, which is given by:

$$v^T A v = (1 - y) p_{b,W}(x) \left[\sum_r q^r_{b,W}(x) v_r \right]^2 - (p_{b,W}(x) - y) \sum_r p^r_{b,W}(x) q^r_{b,W}(x) v_r^2$$

There are two cases to consider:

- *Positive label*: in this case, $v^T A v = (1 - p_{b,W}(x)) \sum_r p^r_{b,W}(x) q^r_{b,W}(x) v_r^2 \geq 0$ for all v and (b, W). In particular, this shows that $\ell_{x,y}$ is *convex* when $y = 1$.

- *Negative label*: for $S \geq 2$, we can always choose a non-zero v satisfying $\sum_r q^r_{b,W}(x)v_r = 0$. In particular, for any such vector we have $v^T Av = -p_{b,W}(x)\sum_r p^r_{b,W}(x)q^r_{b,W}(x)v_r^2 < 0$, proving that $\ell_{x,y}$ is *not convex* when $y = 0$.

□

In light of the above results, we can now discuss which optimization methods are more appropriate to minimize ℓ. First, the loss function's differentiability allows us to apply efficient gradient-based optimization algorithms such as Stochastic Gradient Descent (SGD) [Goodfellow et al., 2016] or Adam [Kingma and Ba, 2015]. Second, the non-convexity of ℓ imposes certain limitations on the quality of solutions found by these methods: we are no longer guaranteed to converge to a global optimum, possibly being trapped at local minima or saddle points. However, we also note that SGD and Adam have been successfully applied to non-convex optimization, especially in the context of Neural Networks [Goodfellow et al., 2016]. In such cases, although these methods may not converge to the global minimum, they still manage to achieve very low values of the loss function and thus are good enough for practical purposes. All in all, we suggest the use of either of these two optimizers.

6.1.2 Feature and Subspace Selection

In this section, we augment the Factorized Linear Model's training procedure by enforcing *sparsity* constraints to the weight matrix W. Sparsity is important for two main reasons:

- *Irrelevant subspace removal*: Irrelevant subspaces are easily identifiable as full-zero rows $w^s = 0$ and, as a result, these rows enable the automatic selection of the number of subspaces S, which is preferred to be small.
- *Subspace feature selection*: Within the remaining subspaces, we can further prune irrelevant features corresponding to zero-valued entries of W, thus enabling a *low-dimensional* factorization structure $(\boldsymbol{A}^1, \ldots, \boldsymbol{A}^S)$ to be computed.

In other words, sparsity improves the model's interpretability since the final classification becomes the simple conjunction of low-dimensional linear classifiers. In what follows, we provide a complete description of the factorization learning procedure, with a step-by-step pseudocode being left to Algorithm 5.

Inducing Sparsity. Our first step is to induce sparsity to the weight matrix while still guaranteeing an accurate final classifier. With this objective in mind, we extend our loss function $\ell(b, W)$ with two penalty terms:

Algorithm 5 Factorization structure calculator

Input: labeled set $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m$, a sample of data points $\mathcal{X}$, an upper bound on number of subspaces $S_{max} \in \mathbb{N}$, thresholds $\epsilon_{zero} \in (0,1)$ and $\epsilon_{prob} \in (0,1)$
Output: A factorization structure compatible with the FLM classifier

$(b^*, W^*) \in \arg\min_{b,W} \ell_{sparse}(b, W)$ for $(b, W) \in \mathbb{R}^{S_{max}} \times \mathbb{R}^{S_{max} \times d}$
relevant_subspaces = [ ]
for $s = 1 \dots S_{max}$ **do**
  if $\|w_s\| > \epsilon_{zero}$ and $\min_{x \in \mathcal{X}} \sigma(b^s + \langle x, w^s \rangle) > 1 - \epsilon_{prob}$ **then**
    relevant_subspaces.append(s)
  end if
end for
factorization = [ ]
for s in relevant_subspaces **do**
  $\boldsymbol{A}^s = \{\}$
  for $i = 1 \dots d$ **do**
    if $|w_i^s| \geq \|w^s\|_1 / d$ **then**
      $\boldsymbol{A}^s = \boldsymbol{A}^s \cup \{i\}$
    end if
  end for
  factorization.append($\boldsymbol{A}^s$)
end for
return factorization

$$\ell_{sparse}(b, W) = \ell(b, W) + \lambda_1 \sum_{s=1}^{S} \|w^s\|_1 + \lambda_2 \sum_{s=1}^{S} \|w^s\|_2 \tag{6.7}$$

where $\|\cdot\|_p$ is the L_p-norm $\|w\|_p = (\sum_i |w_i|^p)^{1/p}$. These penalty terms are based on the *Lasso* and *Group Lasso* penalties, which are well-known for their sparsity-inducing properties:

- **Lasso penalty** [Tibshirani, 1996]: Penalty term based on the L_1 norm $\|w\|_1$. It has the property of inducing sparsity over individual weights, independently pushing each entry of w to zero. In our setting, it induces *sparsity within each subspace*, thus removing irrelevant features.
- **Group Lasso penalty** [Yuan and Lin, 2006]: Penalty term based on the L_2 norm $\sum_s \|w^s\|_2$. It has the property of inducing sparsity over individual subspaces, independently pushing each vector w^s to zero. In our setting, it induces *sparsity over entire subspaces*, thus removing irrelevant subspaces.

In summary, these penalty terms allow us to obtain an accurate solution (b^*, W^*) that is also sparse in both features and subspaces. Additionally, note

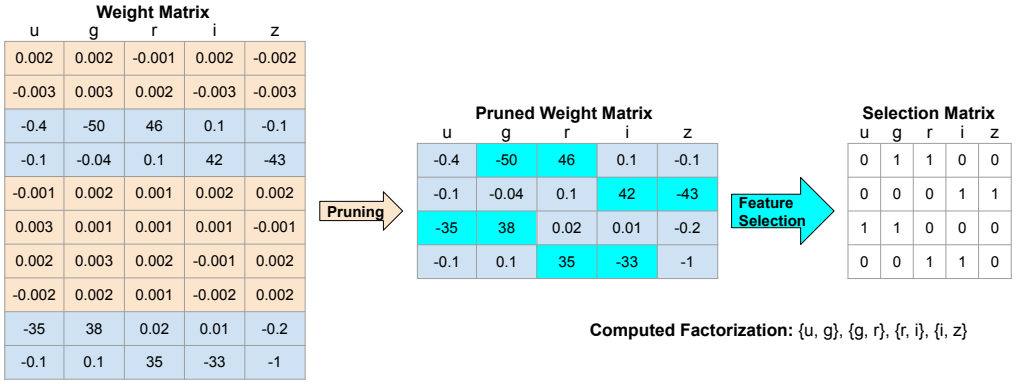


Figure 6.2: Example of our subspace and feature selection procedure over a 10-subspaces penalized FLM trained on SDSS 09. We also display the computed factorization, which is identical to the user's own decomposition. Each row represents a different subspace, and each column a feature. Subspace pruning uses $\epsilon_{zero} = \epsilon_{prob} = 0.01$.

that the bias term b is not penalized since it does not help us identify irrelevant features and subspaces, possibly even hurting the generalization power of our model.

Once (b^*, W^*) has been computed (line 1 in Algorithm 5), we still need an algorithm for pruning the irrelevant subspaces and features from this matrix. Due to numerical issues such as round-off errors and the limited number of optimization iterations, the weights in W^* may be small but not exactly zero, creating a need for more flexible methods in identifying irrelevant entries.

Pruning Algorithm. Let's first consider the irrelevant subspace identification problem described on lines 2–7 in our pseudo-code. Such subspaces can manifest themselves in two ways:

- **Near-zero weight vectors**: if w^s is near zero, this subspace can be safely eliminated. In practice, we check for the condition $\|w^s\| < \epsilon_{zero}$ where ϵ_{zero} is a small positive number.
- **High-margin predictions**: more generally, a subspace s is irrelevant whenever $\sigma(b^s + x^T w^s) \approx 1$ for all $x \in \mathcal{X}$, $\mathcal{X}$ being a large sample of data points. This is because such subspaces have virtually no impact on the final label prediction $p_{b,W}(x)$. More precisely, we eliminate the subspaces such that $\min_{x \in \mathcal{X}} \sigma(b^s + \langle x, w^s \rangle) > 1 - \epsilon_{prob}$, for some small positive ϵ_{prob}.

In Figure 6.2, we can see an example of the pruning procedure in practice. In this case, there are six subspaces full of near-zero weights, and they are all

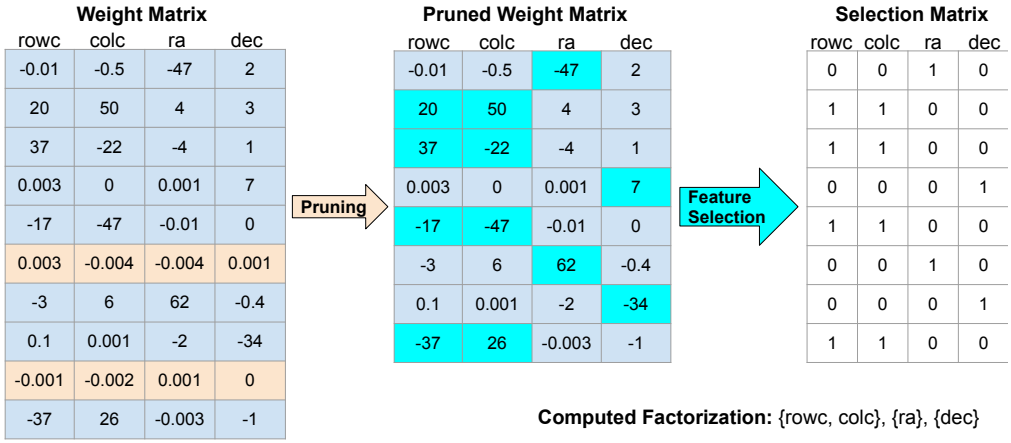


Figure 6.3: Example of our subspace and feature selection procedure over a 10-subspaces penalized FLM trained on SDSS 05. We also display the computed factorization (without duplicates), which is identical to the user's own decomposition. Each row represents a different subspace, and each column a feature. Subspace pruning uses $\epsilon_{zero} = \epsilon_{prob} = 0.01$.

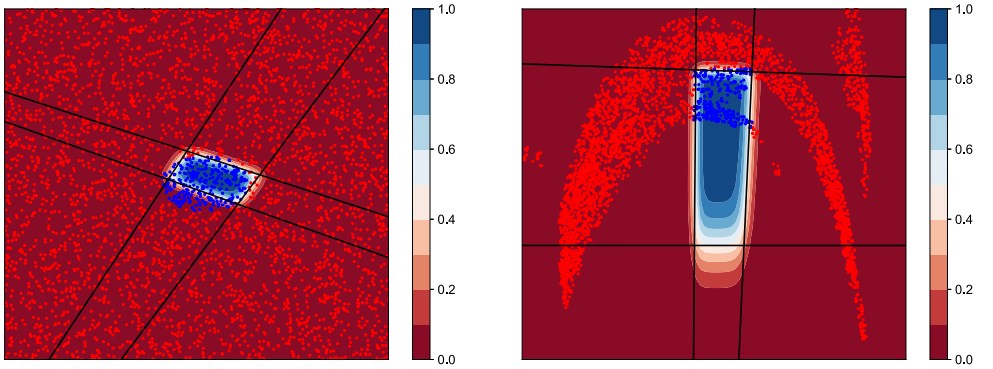
correctly identified by our pruning criteria. In particular, this shows that the Group Lasso penalty is being effective in pushing the weight vectors of irrelevant subspaces to zero.

Once the irrelevant subspaces are removed, we can focus on removing irrelevant features; see lines 10–15 in Algorithm 5 for more details. First, let's assume that each feature is *standardized* to have zero mean and unit variance, which puts all weights w_i^s on the same scale. Without scaling, a particular weight value could be larger or smaller simply because the feature itself spans a set of small or large values; in other words, scaling guarantees that each weight represents the *importance of its feature*. Under this assumption, the feature selection algorithm aims to remove feature i in subspace s whenever its corresponding weight w_i^s is small w.r.t. the other entries in w^s. In particular, we use the following criterion:

$$w_i^s \text{ is relevant} \iff |w_i^s| \geq \frac{\|w^s\|_1}{d} \tag{6.8}$$

The intuition behind this criterium is simple: in the case where all attributes are equally relevant, one would have $|w_i^s| \approx |w_j^s|$ for all i and j or, equivalently, $|w_i^s|/\|w^s\|_1 \approx \frac{1}{d}$ for all i; thus, we can select high-importance attributes by picking those satisfying $|w_i^s|/\|w^s\|_1 \geq \frac{1}{d}$.

In Figure 6.3, we can see an example of the feature selection procedure in practice. Here, our feature selection method can accurately pick the 1 or 2 features per subspace whose weights are several times higher than others. In particular, this shows that the Lasso penalty is being effective in pushing the



(a) {Rowc, Colc} subspace, where user interest has a circular shape.

(b) {Ra, Dec} subspace, where user interest has a rectangular shape.

Figure 6.4: FLM positive class probability distribution over two subspaces for SDSS 05. Dots represent projected data points and their corresponding subspace labels: blue for positive and red for negative. Black lines represent the relevant hyperplanes (b^s, w^s) in each subspace.

weights of irrelevant features to zero in each subspace.

FLM Refinement. Finally, once a factorization structure $(\boldsymbol{A}^1, \ldots, \boldsymbol{A}^S)$ is computed, it can be used to train an unpenalized FLM which is restricted to the selected features. More precisely, we aim to solve the optimization:

$$\underset{(b,W)\in\mathbb{R}^S\times\mathbb{R}^{S\times d}}{\arg\min} \ell(b, W), \text{ s.t. } W_{si} = 0 \text{ for all } i \notin \boldsymbol{A}^s \tag{6.9}$$

The above optimization problem aims to obtain a high-accuracy FLM that respects the factorization structure computed via Algorithm 5. Additionally, we can solve this optimization with minimal modifications to the training procedure: For gradient-based optimization methods such as Adam and SGD, it is simply a matter of setting the gradient of irrelevant features to zero. We also note that the above minimization can be sped up by using the solution to the ℓ_{sparse} minimization as the starting point.

6.1.3 Understanding the Factorized Linear Model

In this section, we aim to build an intuitive understanding of our factorized classifier, exploring which types of user patterns it can accurately learn, as well as any limitations imposed by the linearity assumption.

We begin by observing that FLMs can learn complex patterns in each subspace, not necessarily being restricted to linear ones; in fact, multiple linear

models may be combined in the same subspace, allowing for non-linear decision boundaries to be built. Figure 6.4 illustrates this behavior. There, a penalized FLM has been fit over SDSS 05, a query whose interest pattern can be broken into two subspaces: a circular pattern over {rowc, colc} and a square pattern over {ra, dec}. As we can see, our FLM classifier managed to learn each subspatial interest pattern with a high degree of accuracy. In the first case, despite the non-linearity of the frontier, the FLM still manages to find a good approximation of the decision boundary by combining several hyperplanes. In the second case, the FLM simply places one linear classifier over each side of the square, thus correctly learning the decision boundary.

The above discussion suggests that the FLM should be able to learn user patterns that are the intersection of hyperplanes or, more generally, convex regions. More precisely, we have the following result:

Theorem 6.7 (Convexity of positive region). *Let $(b, W) \in \mathbb{R}^S \times \mathbb{R}^{S \times d}$ be the bias and weight parameters of an FLM model. Then, the set of positively labeled points $Pos(b, W) = \{x \in \mathbb{R}^d : p_{b,W}(x) \geq 0.5\}$ is convex.*

Proof. Through elementary calculations, we have:

$$\begin{aligned} x \in Pos(b, W) &\iff p_{b,W}(x) \geq 0.5 \\ &\iff \prod_{s=1}^{S} \sigma(b^s + \langle x, w^s \rangle) \geq 0.5 \\ &\iff \sum_{s=1}^{S} -\log \sigma(b^s + \langle x, w^s \rangle) \leq \log(2) \end{aligned}$$

This reduces the problem to show that the sublevel set $\{x : f_{b,W}(x) \leq \log(2)\}$ of the function $f_{b,W}(x) = \sum_s -\log \sigma(b^s + \langle x, w^s \rangle)$ is convex. However, this easily follows from a few elementary results in convex analysis:

1. First, we note that $u \mapsto -\log \sigma(u)$ is a convex function over $\mathbb{R}$. This follows from the fact that its second derivative is positive everywhere [Boyd and Vandenberghe, 2004, Section 3.1.4]:

$$\frac{\mathrm{d}^2}{\mathrm{d}u^2} [-\log \sigma(u)] = \sigma(u)(1 - \sigma(u)) > 0, \forall u$$

2. Second, the composition of convex functions with affine mappings always results in convex functions [Boyd and Vandenberghe, 2004, Section 3.2.2], which means that $x \mapsto -\log \sigma(b^s + \langle x, w^s \rangle)$ is also a convex function for any b^s and w^s.

3. Third, convexity is preserved by sums [Boyd and Vandenberghe, 2004, Section 3.2.1], which shows that $f_{b,W}$ is a convex function.

4. Finally, the sublevel sets of convex functions are always convex sets [Boyd and Vandenberghe, 2004, Section 3.1.6], which means that $\{x : f_{b,W}(x) \leq \log(2)\}$ is convex, concluding the proof.

□

In summary, the FLM classifier models the user interest set as a convex object and, in general, should be able to learn convex user interest patterns. However, the opposite is also true, and the FLM should have trouble learning non-convex user interest patterns such as concave shapes and multiple positive regions. Additionally, we note that previous works on factorization [Huang et al., 2018] also worked under similar convexity assumptions, which were deemed a good approximation to real user interest patterns; in fact, by looking at the SDSS and Car data set queries in Appendix D, we can see that the vast majority of queries are combinations of convex predicates.

It is also interesting to notice that the FLM can learn a convex approximation of the user interest much more efficiently than traditional algorithms, especially in high-dimensions. As an example, let $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m$ be a labeled set in $\mathbb{R}^d$, and let's assume we want to approximate the interest region as a convex object. One standard approach to this problem is to compute the convex hull of interest points, but this method scales very poorly with the dimensionality: in fact, state-of-the-art algorithms such as Quickhull [Barber et al., 1996] have a complexity of $\mathcal{O}((sm)^{d/2})$, s being the fraction of positive points in $\mathcal{L}$. In contrast, FLM's optimization can be solved in $\mathcal{O}(ESmd)$ where E is the number of optimization epochs and S the number of subspaces, thus scaling much better with the input dimension.

6.1.4 Factorized Kernel Model

In the previous section, we have seen that the Factorized Linear Model has a few restrictions on the user patterns it can accurately learn. To enhance the generalization power of our algorithm, in this section we will extend it to support *kernels*.

In order to add kernel support to a linear classification model, the usual approach is to rewrite the original model formulation over the feature space. By applying this idea to the FLM, we are led to the following definition:

Definition 6.8 (Factorized Kernel Model (FKM)). *Let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel function and let $\phi : \mathbb{R}^d \to \mathcal{F}$ be the corresponding feature map. A Factorized Kernel Model (FKM) is a probabilistic model for the labeled data (X, Y) on the form:*

$$\mathbb{P}(Y = 1|X = x) = p_{b,\omega}(x) = \prod_{s=1}^{S} \sigma\left(b^s + \langle \phi(x), w^s \rangle_{\mathcal{F}}\right) \tag{6.10}$$

where $b \in \mathbb{R}^s$ is the bias vector and $\omega = (w^1, \ldots, w^S) \in \mathcal{F}^S$ are the feature space weight vectors.

In other words, a Factorized Kernel Model is an FLM with each data point x mapped to $\phi(x)$ and each weight vector w^s replaced with its feature space counterpart. Moreover, note that in the special case $S = 1$ we recover the well-known Kernel Logistic Regression classifier formulation.

Now, let us turn to the problem of training an accurate FKM. By having Equation (6.5) as a basis, we are naturally led to the following loss function:

Definition 6.9 (Cross-entropy loss for FKM). *Let $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{0, 1\}$ be a collection of labeled data points. The cross-entropy loss function for the Factorized Kernel Model is defined as:*

$$\ell_{kernel}(b, \omega) = \frac{1}{m} \sum_{i=1}^{m} -y_i \log p_{b,\omega}(x_i) - (1 - y_i) \log(1 - p_{b,\omega}(x_i)) \tag{6.11}$$

Now that we have a loss function, we turn to the problem of minimizing it:

$$\min_{b \in \mathbb{R}^S, \, \omega \in \mathcal{F}^S} \ell_{kernel}(b, \omega) \tag{6.12}$$

Since $\mathcal{F}$ is usually high-dimensional or even infinite-dimensional, solving the above optimization problem is generally intractable. To counter this dimensionality issue, we rely on the following theoretical result:

Theorem 6.10 (Representer theorem - FKM). *Let's define the linear subspace $U = span(\phi(x_1), \ldots, \phi(x_m)) \subset \mathcal{F}$. Then, the optimization problem (6.12) can be simplified to:*

$$\min_{b \in \mathbb{R}^S, \, \omega \in \mathcal{F}^S} \ell_{kernel}(b, \omega) = \min_{b \in \mathbb{R}^S, \, \alpha \in \mathbb{R}^{S \times m}} \ell_{kernel} \left(b, \sum_{i=1}^{m} \alpha_i^s \phi(x_i) \right)$$

In other words, we can restrict $\omega \in \mathcal{F}^S$ to the finite-dimensional subspace U without any losses.

Proof. By the Orthogonal Decomposition Theorem for Hilbert Spaces, we can write $\mathcal{F} = U \oplus U^\perp$ where $U^\perp$ is the orthogonal complement of U. In particular, this means that for any $w^s \in \mathcal{F}$ there is $\alpha^s \in \mathbb{R}^m$ and $\nu^s \in U^\perp$ satisfying:

$$w^s = \sum_{i=1}^{m} \alpha_i^s \phi(x_i) + \nu^s$$

However, by using the fact that $\langle \phi(x_i), \nu^s \rangle_{\mathcal{F}} = 0$ for all i and s, it is easy to see that $p_{b,\omega}(x_i) = p_{b,\sum_j \alpha_j^s \phi(x_j)}(x_i)$ for all i. In particular, this means that the value of the loss function ℓ_{kernel} remains unchanged if we restrict w^s to vectors on the format $\sum_{i=1}^m \alpha_i^s \phi(x_i)$, which concludes the proof. □

With the above result, we can train a Factorized Kernel Model by solving the following optimization problem:

$$\min_{b \in \mathbb{R}^S,\, \alpha \in \mathbb{R}^{S \times m}} \frac{1}{m} \sum_{i=1}^{m} -y_i \log p_{b,\alpha}(x_i) - (1 - y_i) \log(1 - p_{b,\alpha}(x_i)) \tag{6.13}$$

where $p_{b,\alpha}(x) = \prod_{s=1}^S \sigma\left(b^s + \sum_{i=1}^m \alpha_i^s k(x, x_i)\right)$. Note that, in this last part, we have used the reproducing kernel property $\langle \phi(x), \phi(x') \rangle_{\mathcal{F}} = k(x, x')$.

Additionally, we briefly discuss how to solve the above optimization problem in practice. We start by noticing that $p_{b,\alpha}(x_i) = \prod_{s=1}^S \sigma\left(b^s + \sum_{j=1}^m \alpha_j^s k(x_i, x_j)\right) = \prod_{s=1}^S \sigma\left(b^s + \langle K_i, \alpha^s \rangle\right)$, where K_i is the i-th row of the $m \times m$ kernel matrix $K_{ij} = k(x_i, x_j)$. Thus, the optimization procedure remains the same as in the linear case, being just a matter of replacing each x_i with its kernelized version K_i. Another point to consider is the memory consumption of the algorithm: storing the kernel matrix occupies $\mathcal{O}(m^2)$ space, which can be prohibitive for large data sets. One way to alleviate this problem is to use *kernel approximation* methods such as the Nystroem approximation [Williams and Seeger, 2001]; such methods can approximate the full $m \times m$ kernel matrix by a $m \times q$ matrix with $q \ll m$ and minimal performance loss.

FKM Refinement. To conclude, we notice that the FKM also supports a factorization structure $(\boldsymbol{A}^1, \ldots, \boldsymbol{A}^S)$ if available; for example, it could be computed via Algorithm 5. In such cases, one simply has to replace each $\langle K_i, \alpha^s \rangle$ with $\langle K_i^s, \alpha^s \rangle$, where K_i^s is the i-th row of the kernel matrix $K^s = [k(x_i^s, x_j^s)]$; in other words, it is just a matter of restricting the kernel computations to each subspace. In addition, note that due to the high generalization power of kernel classifiers, we can also simplify the factorization structure by excluding redundant subspaces. Take Figure 6.4(a) as an example: while four linear classifiers were necessary to learn a circular pattern in this subspace, one single kernel classifier is enough to learn this shape. More precisely, we propose the removal of any subspace s for which:

$$\exists s' \neq s \text{ such that } \boldsymbol{A}^s \subseteq \boldsymbol{A}^{s'} \tag{6.14}$$

In doing so, we can reduce the number of parameters to fit with minimal impact on performance.

Comparison to Multiple Kernels. We also briefly discuss how our FKM classifier compares against Multiple Kernel Learning [Gönen and Alpaydın, 2011], a piece of related work in the ML literature. Multiple Kernel Learning methods aim to generate a kernel function customized to the learning problem at hand by composing multiple kernel functions together. For example, one popular strategy [Tanabe et al., 2008, Qiu and Lane, 2009] is to fit a linear combination of kernels $k^* = \sum_{i=1}^{n} \beta_i k_i$, where the parameters β_i must now be inferred as part of the loss-minimization routine. Compared to our FKM algorithm, we can draw the following conclusions:

- *Similarities*: For both methods, the final prediction is obtained by combining multiple kernels.
- *Differences*: Both approaches differ in terms of how to combine available kernels. On one hand, FKM treats kernel functions as independent classifiers that, combined, derive the user's interest region. On the other hand, Multiple Kernel methods still make their final prediction through a single kernel function, which is obtained through the combination of multiple kernels.

6.1.5 Dealing with Categorical Variables

One last point to consider is how the FLM classifier deals with categorical variables. Being a composition of linear models, any non-numerical feature must first be encoded into a numerical representation. The trade-offs between different encoding methods have been extensively studied in the literature [Chiquet et al., 2016], and its choice depends on the particular properties of the categorical feature in question.

However, the nature of the encoding process also poses a problem when computing a factorization structure via Algorithm 5. Since the encoded columns are merely a numerical proxy to the underlying categorical features, they should be treated as a *single group of features* instead of independent attributes. To achieve this property, we propose two small modifications:

1. *Group penalization of encoded features*: when training a sparse model, the Lasso penalty tends to select features independently from each other, which results in only a subset of encoded features being chosen. In order to induce a grouped selection which either picks the whole set of encoded features or not, we replace the Lasso penalty with Group Lasso over groups of encoded features [Chiquet et al., 2016]. More precisely, let $\xi : \boldsymbol{A}_{enc} \rightarrow \boldsymbol{A}_{ori}$ be the function mapping each encoded feature to the original attribute it is derived from. Then, ℓ_{sparse} should be modified as follows:

$$\ell_{sparse}(b, W) = \ell(b, W) + \sum_{s=1}^{S} \sum_{a \in \boldsymbol{A}_{ori}} \|w^s_{\xi^{-1}(a)}\|_2 + \sum_{s=1}^{S} \|w^s\|_2 \tag{6.15}$$

In particular, note that the above definition is reduced to the usual Lasso for numerical features since in these cases $\xi^{-1}(a) = \{a\}$ (no encoding).

2. *Compute importance of a categorical attribute as a whole*: additionally, the feature selection criteria of Equation (6.8) is modified to select $a \in \boldsymbol{A}_{ori}$ according to the following rule:

$$a \in \boldsymbol{A}_{ori} \text{ is relevant } \iff \sum_{i \in \xi^{-1}(a)} |w_i| \geq \frac{\|w\|_1}{|\boldsymbol{A}_{ori}|} \tag{6.16}$$

In other words, we model the *importance* of a categorical feature as the sum of the importance of individual encoded features. Once again, we note that this new criterion reduces to the original one for numerical attributes.

6.2 The Active Learning Scenario

In this section, we discuss how the Factorized Linear Model can be applied to the active learning scenario. Intuitively speaking, an FLM-based exploration strategy can provide a couple of benefits when compared to pure active learning methods such as OptVS (presented in Chapter 4), or explicitly factorized algorithms such as the Factorized Version Space (Chapter 5) and DSM [Huang et al., 2018]:

- *Improved performance*: We have seen in Section 5.7 that factorized strategies often outperform pure active learning algorithms by a large margin, which suggests that similar improvements could be achieved by an FLM-based exploration strategy.
- *No factorization input from the user*: The FLM can automatically learn the factorization structure from the labeled data, not requiring any extra information from the user apart from the "interesting / not interesting" labels for each data point.

In summary, an FLM-based strategy is a middle ground between the explicitly factorized and pure active learning approaches, enjoying the improved accuracy of factorization while requiring no extra input from the user. Because of this, we also refer to this strategy as an *automatically factorized* algorithm.

6.2.1 Uncertainty Sampling for FLM

To begin, since the FLM is a probabilistic classifier, we propose a data exploration strategy based on the Uncertainty Sampling criterium [Settles, 2012], which we define below:

Definition 6.11 (Uncertainty Sampling strategy - FLM). *Let $\mathcal{L}$ and $\mathcal{U}$ be the sets of labeled and unlabeled points at any iteration of the data exploration routine. Also, let (b, W) be the bias and weight parameters of an FLM trained over $\mathcal{L}$. Then, the Uncertainty Sampling strategy selects next for sampling the unlabeled point for which the classifier is most uncertain:*

$$\arg\min_{x \in \mathcal{U}} |p_{b,W}(x) - 0.5| \tag{6.17}$$

Uncertainty Sampling-based approaches have the advantage of being very simple to implement and efficient to run, but its simplicity comes with a cost: such methods can lead to myopic selection criteria [Settles, 2012], resulting in slower convergence rates when compared to version space algorithms and other more sophisticated approaches. However, complex methods usually are very time-consuming in practice, which makes them incompatible with the interactive data exploration scenario. Let us consider version space (VS) algorithms, for example. Effective VS approaches usually require sampling from the version space, which is hard to do efficiently in practice. In particular, for our VS-based algorithm, OptVS, we had to include several sampling optimizations to make it run under interactive performance; see Chapter 4 for more details. In the case of the FLM, such an efficient sampling procedure is even harder to achieve due to the following result:

Theorem 6.12. *Let $\mathcal{L} \subset \mathbb{R}^d \times \{-1, 1\}$ be a labeled set which is **not** linearly separable. Then, the version space of an FLM classifier:*

$$\mathcal{V}_{FLM} = \{(b, W) \in \mathbb{R}^S \times \mathbb{R}^{S \times d} : y(p_{b,W}(x) - 0.5) \geq 0 \text{ for } (x, y) \in \mathcal{L}\} \tag{6.18}$$

is either an empty or a non-convex set.

Proof. We prove the contrapositive of the theorem: assuming that $\mathcal{V}_{FLM}$ is a non-empty convex set, we show that $\mathcal{L}$ can be separated by a linear classifier.

With the above in mind, let (b, W) be any point belonging to the version space. By Theorem 6.6, we know that $p_{b,W}(x)$ remains unchanged by permutating the order of subspaces; in particular, it guarantees that each $(b^{(i)}, W^{(i)})$ of the

form:

$$b^{(i)} = \begin{bmatrix} b_i \\ b_{i+1} \\ \dots \\ b_S \\ b_1 \\ \dots \\ b_{i-1} \end{bmatrix}_{S\times 1}, \qquad W^{(i)} = \begin{bmatrix} w_i \\ w_{i+1} \\ \dots \\ w_S \\ w_1 \\ \dots \\ w_{i-1} \end{bmatrix}_{S\times d}$$

also belongs to the version space. However, since we are assuming $\mathcal{V}_{FLM}$ to be a convex set, the average point $(\bar{b}, \bar{W}) = \left(\frac{1}{S}\sum_{i=1}^{S} b^{(i)}, \frac{1}{S}\sum_{i=1}^{S} W^{(i)}\right)$ must also belong to $\mathcal{V}_{FLM}$. In particular, it is easy to see that $(\bar{b}, \bar{W})$ satisfies:

$$\frac{1}{S}\sum_{i=1}^{S} b^{(i)} = \frac{1}{S}\begin{bmatrix} b_1 & + & b_2 & + & \dots & + & b_{S-1} & + & b_S \\ b_2 & + & b_3 & + & \dots & + & b_S & + & b_1 \\ & & & & \vdots & & & & \\ b_S & + & b_1 & + & \dots & + & b_{S-2} & + & b_{S-1} \end{bmatrix}_{S\times 1} = \begin{bmatrix} b^* \\ \vdots \\ b^* \end{bmatrix}_{S\times 1}$$

$$\frac{1}{S}\sum_{i=1}^{S} W^{(i)} = \frac{1}{S}\begin{bmatrix} w_1 & + & w_2 & + & \dots & + & w_{S-1} & + & w_S \\ w_2 & + & w_3 & + & \dots & + & w_S & + & w_1 \\ & & & & \vdots & & & & \\ w_S & + & w_1 & + & \dots & + & w_{S-2} & + & w_{S-1} \end{bmatrix}_{S\times d} = \begin{bmatrix} w^* \\ \vdots \\ w^* \end{bmatrix}_{S\times d}$$

with $b^* = \frac{1}{S}\sum_{i=1}^{S} b_i$ and $w^* = \frac{1}{S}\sum_{i=1}^{S} w_i$; in other words, all the subspaces have identical parameters (b^*, w^*).

Now, since $(\bar{b}, \bar{W})$ belongs to the version space, it must correctly label all points in $\mathcal{L}$, that is:

$$\begin{aligned} &\forall (x,y) \in \mathcal{L},\ y\left(p_{\bar{b},\bar{W}}(x) - 0.5\right) > 0 \\ &\forall (x,y) \in \mathcal{L},\ y\left(\sigma(b^* + \langle x, w^*\rangle)^S - 0.5\right) > 0 \\ &\forall (x,y) \in \mathcal{L},\ y\left(b^* + \langle x, w^*\rangle - \sigma^{-1}(0.5^{1/S})\right) > 0 \end{aligned}$$

which proves that $\mathcal{L}$ is linearly separable by the hyperplane $(b^* - \sigma^{-1}(0.5^{1/S}), w^*)$ □

Compared to our OptVS algorithm in Chapter 4, the non-convexity of the version space forbids us from using efficient sampling techniques such as the hit-and-run algorithm [Lovász, 1999], including the several optimizations we devised for improving the time cost.

6.2.2 Swapping Algorithm for FLM

With the above considerations, we now focus on countering the slow convergence of the Uncertainty Sampling strategy (6.17). In this regard, we propose a novel

Algorithm 6 The Swapping Algorithm

Input: labeled set $\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{t}$, unlabeled set $\mathcal{U}$, the swap limit N_{swap}, an active learner $\mathcal{A}$, sample size m, number of subspaces S, Lasso penalty $\lambda_1 > 0$, Group Lasso penalty $\lambda_2 > 0$

Output: The next data point to be labeled

1: **if** $t < N_{swap}$ **then**
2: $\quad \mathcal{A}.fit(\mathcal{L})$
3: $\quad$ **return** $\mathcal{A}.get\_next\_to\_label(\mathcal{U})$
4: **else if** $t = N_{swap}$ **then**
5: $\quad \{\tilde{x}_j\}_{j=1}^{m} \leftarrow subsample(\mathcal{U}, m)$
6: $\quad \{\tilde{y}_j\}_{j=1}^{m} \leftarrow \mathcal{A}.predict(\{\tilde{x}_j\})$
7: $\quad \ell(b, W) \leftarrow \frac{1}{m} \sum_{j=1}^{m} \ell_{\tilde{x}_j, \tilde{y}_j}(b, W)$
8: $\quad (b_t, W_t) \in \arg\min_{b,W} \ell(b, W)$
9: $\quad$ **return** $\arg\min_{x \in \mathcal{U}} |p_{b_t, W_t}(x) - 0.5|$
10: **else**
11: $\quad \ell_{sparse}(b, W) \leftarrow \frac{1}{t} \sum_{i=1}^{t} \ell_{x_i, y_i}(b, W) + \lambda_1 \sum_{s=1}^{S} \|w^s\|_1 + \lambda_2 \sum_{s=1}^{S} \|w^s\|_2$
12: $\quad (b_t, W_t) \in \arg\min_{b,W} \ell_{sparse}(b, W)$, using (b_{t-1}, W_{t-1}) as starting point
13: $\quad$ **return** $\arg\min_{x \in \mathcal{U}} |p_{b_t, W_t}(x) - 0.5|$
14: **end if**

data exploration strategy called the *Swapping Algorithm*, which is fully described in Algorithm 6. This method divides the exploration procedure into three main phases:

1. **Active Learning phase** (lines 1–3): Initially, the algorithm aims to obtain a moderately accurate classifier within as few labeled examples as possible. More precisely, for the initial N_{fast} labeling iterations, we simply employ any non-factorized active learning algorithm $\mathcal{A}$ known to converge quickly in practice; for example, we could use our OptVS.

2. **Swapping phase** (lines 4–9): Once N_{fast} iterations are reached, we train an FLM over $\{(\tilde{x}_j\, \tilde{y}_j)\}$, where $\{\tilde{x}_j\}$, is a sample from the data and $\tilde{y}_j$ is $\mathcal{A}$'s label prediction for $\tilde{x}_j$. The objective here is to obtain an FLM mimicking the active learner $\mathcal{A}$, thus preparing for the change of classifiers in subsequent iterations. Also, note that we do not include penalty terms in training to avoid compromising the FLM's accuracy.

3. **Factorized phase** (lines 10–13): For the remaining labeling iterations, we further improve the FLM classifier by picking points according to the Uncertainty Sampling criterion (6.17). However, to ensure a smooth transition between learning algorithms, special care must be taken when training the FLM: since the initial N_{fast} labeled points were chosen by $\mathcal{A}$, an FLM trained over this labeled data will not generalize well over the entire data

set. To overcome this problem, the FLM's training procedure is enhanced with two special modifications:

- *Incremental training*: Let (b_t, W_t) be the optimal bias and weight parameters computed at iteration t, with $(b_{N_{fast}}, W_{N_{fast}})$ defined as the Swapping phase model parameters. Then, at iteration $t+1$ we use (b_t, W_t) as starting point for the optimization routine minimizing ℓ_{sparse} (line 12). In this way, the FLM classifier changes incrementally from one iteration to the other, slowly deviating from the Swapping phase solution as new points are labeled.
- *Max iterations scheduling*: To ensure a more controlled transition from the Swapping phase model, we limit the number of optimization iterations when training the FLM, gradually increasing this limit until a maximum value is reached. In particular, we propose an *exponential scheduler* on the form:

$$maxIterScheduler(t) = \min\left(m_2,\, m_1 \cdot 2^{(t-N_{fast}-1)/r}\right) \tag{6.19}$$

  where m_1 and m_2 are the minimum and the maximum number of iterations, and r is the exponential decay factor. As an example, in the case $m_1 = 10$, $m_2 = 1000$, and $r = 5$, one starts with 10 optimization iterations, doubling this value every 5 labeled points until 1000 iterations are reached.

To conclude, we also note that penalty terms are included in this phase since they simplify our classification model by removing irrelevant features and subspaces.

In summary, the three-step method avoids the slow convergence of the Uncertainty Sampling strategy in (6.17) by initially running a fast converging active learner, incrementally switching to the FLM-based strategy after several iterations. By appropriately choosing $\mathcal{A}$ and N_{fast}, we can effectively skip the initial slow-convergence phase, and thus obtain a high-accuracy FLM within fewer iterations than simply running the Uncertainty Sampling strategy from the beginning.

6.2.3 Switching to an Explicitly Factorized Algorithm

In the previous section, we have seen how to effectively apply the FLM in the active learning scenario. One natural question that arises, however, is whether the FLM could learn a relevant factorization and enable the switch to an explicitly factorized algorithm such as the Factorized Version Space or DSM [Huang et al., 2018]. In theory, as more points are labeled and the FLM classifier becomes increasingly more accurate, a relevant factorization structure could be computed via Algorithm 5 and then proposed to the user. At this point, the user could either

accept this factorization if it matches his decision-making process or continue labeling until a good factorization appears. However, such a conceptual approach faces several challenges in practice that are not easily surmounted:

- **Uniqueness of factorization**: First, there is the problem of the uniqueness of a factorization structure. Depending on several factors such as the number of subspaces and the amount of penalty used, different factorizations may be computed as being more relevant for the particular labeled data distribution in question. This can cause two problems: first, the computed factorizations may be very different from the user's decision-making process, with the user being unable to provide subspace labels; second, from a scientific perspective, we cannot easily simulate how this switch would perform in practice since it could only be done when the exact user factorization is computed, which is generally not the case.

- **Lack of subspace labels**: All data points labeled before the factorization selection are missing their subspace labels, and thus cannot be used for training the explicitly factorized model. This can lead to two scenarios: in one case, we simply ignore the initial labeled points and start exploration from scratch, but this time with the factorization knowledge; in the second case, we can use the FLM model to predict the partial labels for previous points, thus enabling the use of the entire labeled data. However, both approaches are not ideal in practice: in the first case, the user has effectively labeled dozens of points only to learn a factorization, and it may take a few more dozens to match the previous algorithm's performance; in the second case, active learning algorithms are known to be very sensitive to label noise, which even in small amounts can have a significant impact in performance. This is particularly problematic for version space algorithms, where noisy labels may lead to an empty version space.

- **Model swapping hinders performance**: As we discussed in the previous section, swapping classification models during the exploration process can hurt performance and must be done slowly. For classifiers such as the FLM, we can incrementally train the algorithm and let it slowly adapt as more points are labeled. However, sampling-based algorithms such as the Majority Vote classifier or the convex-hull-based approaches of DSM [Huang et al., 2018] do not easily support such forms of incremental training and must cope with a significant drop in performance at least for early iterations after the explicitly factorized switch.

In conclusion, switching to an explicitly factorized algorithm can be a promising avenue of research for its performance benefits, but it still faces several non-trivial technical challenges which go beyond the scope of this thesis and thus will be left to future work.

6.3 Evaluation in the Batch Learning Setting

In this first experimental section, we aim to observe how our Factorized Linear Model (FLM) and its variants perform in the *batch learning* setting, in which classification algorithms have a large pool of labeled data at their disposal for training. We also consider how our techniques compare to other classifiers in terms of accuracy, training time, and interpretability of the final model.

Experimental Setup

Data sets and user interest patterns: As in the previous chapters, we evaluate our algorithms over the SDSS and Car data sets; see Section 4.4 and Appendix D for a complete description of the data and user interest patterns. More precisely, we compare all algorithms over the 7 high-dimensional SDSS queries, 05–011, and all the 18 Car queries; SDSS queries 01–04 are ignored due to their low-dimensionality (2D only) and thus lack of a proper factorization. This results in a total of 25 labeled data sets (taken as ground truth) for performance evaluation.

Model selection criteria: Since the SDSS and Car data sets are very distinct, we also employ different model selection criteria to best evaluate and compare different algorithms. First, since SDSS is a large data set with nearly 1.8 million points, we adopt a simple 50% train / 50% test random split for evaluation, comparing models in terms of their classification accuracy over the test set. As for the Car data set, the situation is more complicated: not only it is relatively small with only 5622 data points, but the user patterns are so selective that they amount to no more than a few dozens of positive points. To obtain a more precise estimate of a model's performance, we adopt a Repeated Stratified k-Fold Cross Validation procedure with $k = 5$ and 20 repeats. Stratification guarantees that each fold has a reasonable amount of positive points, while repetitions decrease the variance of the estimated performance metric. Here, we compare algorithms based on their average accuracy across all train-test splits, considering a difference significant whenever values differ by more than one standard deviation.

Algorithms: We consider five different versions of our Factorized Linear Model algorithm, each one corresponding to different trade-offs in terms of *generalization power* and *interpretability*:

- *Unpenalized FLM*: trained by minimizing the cross-entropy loss ℓ of Equation 6.5.
- *Penalized FLM*: trained by minimizing the sparse cross-entropy loss ℓ_{sparse} of Equation 6.7.

- *Refined FLM*: trained by minimizing the cross-entropy loss under fixed factorization of Equation 6.9. A factorization is computed via Algorithm 5.
- *FKM*: the kernelized version of our FLM algorithm, obtained by solving the optimization problem of Equation 6.13.
- *Refined FKM*: FKM trained under a fixed factorization, as described at the end of Section 6.1.4. Once again, factorization is first computed via Algorithm 5.

We also compare our techniques against two other classification algorithms:

- *Support Vector Machines (SVM)*: SVM is included here as a baseline, being a popular algorithm with high predictive power and is also the basis for previously considered active learning strategies such as Simple Margin [Tong and Koller, 2001].
- *VIPR*[1] [Fiterau and Dubrawski, 2012]: VIPR is a PRC method that can provide interpretable results as a set of low-dimensional projections, similar to the idea behind factorization. We also note that only VIPR's *greedy* selection strategy will be considered in the experiments since the *adaptive* selection strategy is too memory-intensive for our data sets.

Hyper-parameter tuning: Hyper-parameters were selected through the same procedure of Section 4.4, that is, for each algorithm, we choose the parameters that provide the best performance over the entire query population.

First, SVM uses the RBF kernel with default γ and $C = 10^5$. Second, VIPR uses the *greedy* method to retrieve the top P projections over each data set: for SDSS, it selects $P = 2$ with a selection parameter of $K = 50$, while over Car we use $P = 1$ and $K = 3$. Projections were restricted to be two or three-dimensional, and a kNN classifier ($k = 5$) is trained over each partition for classification. Finally, FLM parameters depend on the variant and the data set considered. In terms of number of subspaces, Unpenalized and Penalized FLM assume $S = 10$ while FKM uses $S = 3$. Penalty terms for the penalized version are $\lambda_1 = \lambda_2 = 10^{-5}$ for all cases except SDSS 05, for which we used $\lambda_1 = \lambda_2 = 10^{-7}$. All optimizations are solved via Adam [Kingma and Ba, 2015] with default parameters: For SDSS queries, the optimizer is run for 200 epochs with a batch size of 200 points and exponentially decaying step sizes, starting at 0.05 and decaying by 0.9 every 20 epochs; as for Car, we use full batches and a fixed step size of 0.5 for 2000 iterations. Kernel algorithms are also trained over these same conditions, with a doubled number of iterations for Car queries. Finally, the kernel-based variants assume a RBF kernel with default γ, employing

[1] VIPR code is publically available at `https://github.com/inafiterau/VIPR`

Query	FLM (Unpenalized)	FLM (Penalized)	FLM (Refined)	FKM	FKM (Refined)	SVM	VIPR
05	82	67	79	72	**88**	80	73
06	5	0	0	3	0	**18**	10
07	63	61	47	65	48	**90**	72
08	**100**	96	94	98	97	98	88
09	99	96	**100**	94	99	92	75
10	**96**	79	86	92	89	89	41
11	**86**	73	84	**86**	**86**	83	71

Table 6.1: Test F-score over a 50% / 50% train-test random split for SDSS queries. Columns shaded in grey represent non-interpretable models (no feature decomposition), and bold values correspond to the highest value for each query.

the Nystroem approximation [Williams and Seeger, 2001] with 100 components for computing the kernel matrix.

Evaluating Non-Interpretable Models

We start by evaluating the non-interpretable models against each other; more precisely, we compare our Unpenalized FLM and FKM classifiers against SVM. Below, we summarize the general behaviors of each algorithm, deferring the complete set of results on classification accuracy to Tables 6.1 and 6.2 and the time measurements to Table 6.3.

First, the Unpenalized FLM is the most accurate classifier in general, presenting similar or better performance than all other algorithms in 20 out of 25 labeled data sets. In particular, we note that this classifier outperforms the remaining FLM-based algorithms in all cases but two and, consequently, can be considered to upper bound them. We also observe a poor performance for the non-convex user patterns SDSS 06 and 07, achieving an abysmal 5% accuracy for 06. This behavior can be explained by the convexity results of Theorem 6.12, which suggests that the FLM should have trouble learning non-convex user interest patterns. However, we also note that our algorithm may still be able to cope with small degrees of non-convexity: SDSS 11 is a query composed of five subspaces with two being non-convex, but our Unpenalized FLM still manages to achieve the highest accuracy.

Next, we examine our kernelized algorithm. In general, FKM presents good classification accuracy, achieving comparable performance to Unpenalized FLM in 17 queries and larger than 85% F-score in more than half of all cases. Compared to SVM, we observed similar or better performance in 21 out of the 25 results, which indicates that FKM can provide an edge in terms of generalization power: comparatively, while the SVM attempts to fit a single kernel model with

Query	FLM (Unpenalized)	FLM (Penalized)	FLM (Refined)	FKM	FKM (Refined)	SVM	VIPR
01	**96 ± 1**	90 ± 1	88 ± 2	90 ± 1	34 ± 4	88 ± 1	69 ± 1
02	78 ± 2	**84 ± 2**	**87 ± 2**	83 ± 2	72 ± 4	67 ± 3	84 ± 0
03	**96 ± 1**	**94 ± 1**	38 ± 5	**96 ± 1**	58 ± 4	**96 ± 1**	79 ± 1
04	**86 ± 1**	65 ± 3	78 ± 3	81 ± 3	75 ± 3	80 ± 2	25 ± 3
05	**97 ± 1**	90 ± 1	**96 ± 1**	79 ± 3	58 ± 5	88 ± 1	37 ± 4
06	75 ± 4	73 ± 4	71 ± 4	73 ± 4	38 ± 4	72 ± 4	**89 ± 1**
07	**94 ± 1**	**93 ± 1**	**94 ± 1**	**92 ± 1**	35 ± 4	90 ± 1	88 ± 0
08	**100 ± 0**	**100 ± 0**	**100 ± 0**	**100 ± 0**	**100 ± 0**	**100 ± 0**	**100 ± 0**
09	**97 ± 1**	**97 ± 1**	93 ± 2	**97 ± 1**	**96 ± 1**	**96 ± 1**	**97 ± 0**
10	**98 ± 0**	94 ± 1	95 ± 2	88 ± 1	53 ± 5	85 ± 1	0 ± 0
11	**83 ± 2**	**83 ± 2**	**87 ± 2**	74 ± 3	75 ± 4	**80 ± 2**	53 ± 2
12	91 ± 2	87 ± 2	**98 ± 1**	89 ± 2	**95 ± 2**	76 ± 3	69 ± 1
13	**89 ± 2**	**90 ± 2**	83 ± 3	**90 ± 2**	78 ± 3	80 ± 2	15 ± 2
14	83 ± 2	69 ± 3	73 ± 3	80 ± 3	61 ± 3	84 ± 2	**88 ± 1**
15	**97 ± 0**	**98 ± 0**	94 ± 1	**97 ± 0**	63 ± 3	93 ± 1	24 ± 1
16	**100 ± 0**	98 ± 1	76 ± 3	**99 ± 0**	77 ± 3	91 ± 1	95 ± 1
17	**71 ± 2**	65 ± 3	40 ± 4	51 ± 4	13 ± 3	43 ± 3	10 ± 2
18	**95 ± 1**	**96 ± 1**	90 ± 2	**95 ± 1**	84 ± 3	88 ± 2	77 ± 1

Table 6.2: Average test F-score and mean standard deviation of a 20 repeated, 5-fold stratified cross-validation for all Car queries. Columns shaded in grey represent non-interpretable models (no feature decomposition), and bold values correspond to the highest test scores for each query (within 1 SD).

a large margin property, our FKM attempts to combine multiple kernel classifiers together, thus improving its learning capabilities.

Finally, we also compare our algorithms in terms of training time over large data sets, as shown in Table 6.3. First, we note that both FLM and FKM present very similar running times with FKM being slightly slower; this is expected since FKM's training algorithm is identical to FLM's, but uses the (approximated) kernel matrix as input instead of the data matrix. In contrast, SVM is the slowest running algorithm in all cases but one, taking up to several hours in a couple of cases. We also briefly comment on the training time stability among different queries: FLM and FKM times change very little across all queries because they use identical optimization parameters such as batch size and the number of iterations. On the other hand, SVM presents highly variable training times ranging from a few seconds to several hours, which is mainly due to its convex optimization procedure taking longer to detect convergence.

Comparing Interpretable Techniques

Next, we compare our interpretable models, Penalized FLM and its refined variants, against VIPR [Fiterau and Dubrawski, 2012]. These algorithms are deemed

Query	FLM (Unpenalized)	FLM (Penalized)	FLM (Refined)	FKM	FKM (Refined)	SVM	VIPR
05	3.0	3.8	6.6	3.3	13.5	**0.4**	0.5
06	3.0	3.8	6.2	3.2	9.3	16.1	**2.2**
07	3.1	3.8	6.5	3.6	9.1	1158.2	**0.6**
08	**3.1**	4.2	6.8	3.4	9.6	45.7	3.3
09	3.0	3.7	6.3	3.2	11.4	102.9	**1.2**
10	2.9	3.8	6.4	3.2	11.4	10.4	**1.2**
11	2.9	4.1	6.4	3.2	10.8	3.5	**1.2**

Table 6.3: Training time in minutes over SDSS queries. Fastest times for each query are in bold.

interpretable because they decompose the user interest pattern as a set of low-dimensional classifications, thus providing the practitioner with a better intuition about how the predictions are made.

First, let us consider our Penalized FLM. When compared to its Unpenalized version, the Penalized FLM's accuracy is usually lower, achieving similar or better performance only over half the Car queries. A decrease in performance is not surprising since the additional penalty terms in ℓ_{sparse} force a compromise between accuracy and sparsity. However, if we only consider interpretable models, this classifier is often the best performing one, managing to be the most accurate for 17 out of the 25 queries.

Second, we turn our attention to the refined FLM and FKM models, which employ the factorization structure computed via Algorithm 5. In general, these algorithms present mixed results: on one hand, the refined algorithms are better or equivalent to Penalized FLM in half the considered cases, sometimes even matching the Unpenalized FLM performance; however, they are also observed to perform very poorly sometimes, with FKM achieving less than 70% accuracy in almost half of cases. The problem with the refined approaches is that their success is tied to the quality of the computed factorization structure. For example, whenever the Penalized FLM performs very poorly, as in SDSS 06 and 07, the refined methods will receive a suboptimal factorization, and its performance will suffer.

Lastly, we observe that VIPR does not fare well in practice, being the worst performing algorithm in half the considered cases. In particular, VIPR can only surpass our Penalized FLM in four queries, two of them being the non-convex SDSS queries previously discussed. This difference in performance suggests that VIPR cannot capture the user interest pattern as precisely as FLM, which can be explained by the different assumptions under which each model operates: FLM models the user interest pattern as a *conjunction* of low-dimensional convex objects; VIPR, on the other hand, is better suited for *disjunctive* interest patterns,

where the labeled set can be divided into disjoint subsets that can be easily classified with just a few attributes. In fact, since almost all SDSS and Car patterns are conjunctions of simple predicates, our FLM-based models can easily capture the user interest region while VIPR may have trouble finding a single subset of attributes where the positive and negative classes are well separated.

In terms of training time, VIPR is the fastest algorithm for all queries but two, and even in these cases, its training time is not worse by more than a couple of seconds from the best. This fast speed is due to its internal kNN classification model, which is very efficient to train. Additionally, we observe that Penalized FLM is slightly lower than its Unpenalized version due to the extra penalty term gradient to compute. Finally, we note that the refined methods are among the slowest ones, taking around two to three times more time to train than other FLM versions. This difference is mainly caused by the factorization calculation of Algorithm 5 that requires training a Penalized FLM.

Summary of Batch Experimental Results

From the above results, we can draw the following main conclusions:

- *Evaluation of non-interpretable models*: Our Unpenalized FLM is the most accurate classifier we observed, outperforming other methods in 20 out of 25 user patterns. Besides, its kernelized version also achieved high accuracy overall, matching SVM's performance in more than 80% of cases.
- *Evaluation of interpretable models*: Our Penalized FLM can decompose the user interest pattern as a sequence of low-dimensional linear predictions while still maintaining high classification accuracy overall, outperforming other interpretable techniques in nearly 70% of cases. Additionally, the refined FLM and FKM variants can boost performance when an accurate factorization structure is computed.
- *Training time*: Our proposed algorithms train very fast, taking at most a few minutes to compute an accurate model over a large data set of nearly one million points. Comparatively, SVM can take up to several hours in some of the considered cases.
- *Convexity assumption*: We also note that FLM-based strategies may not be able to cope with non-convex user patterns but should still be able to perform reasonably well provided that the user pattern is approximately convex in shape.
- *Recommendation for practical use*: If we had to recommend one FLM variant for practical use, it would be the Penalized FLM. Compared to SVM, this method was shown to approximate or improve its accuracy in 70% of

cases while having the extra advantages of scaling to large data sets and being interpretable. Compared to VIPR, we observed that the Penalized FLM offers better accuracy in 80% of cases at a modest increase in training time, with both models being interpretable.

6.4 Evaluation in the Active Learning Setting

To conclude this chapter, we return to the Interactive Data Exploration scenario, verifying how automatically factorized strategies perform against the active learners introduced in previous chapters, including explicitly factorized algorithms. Once again, we compare the different methods in terms of their convergence speed and running time.

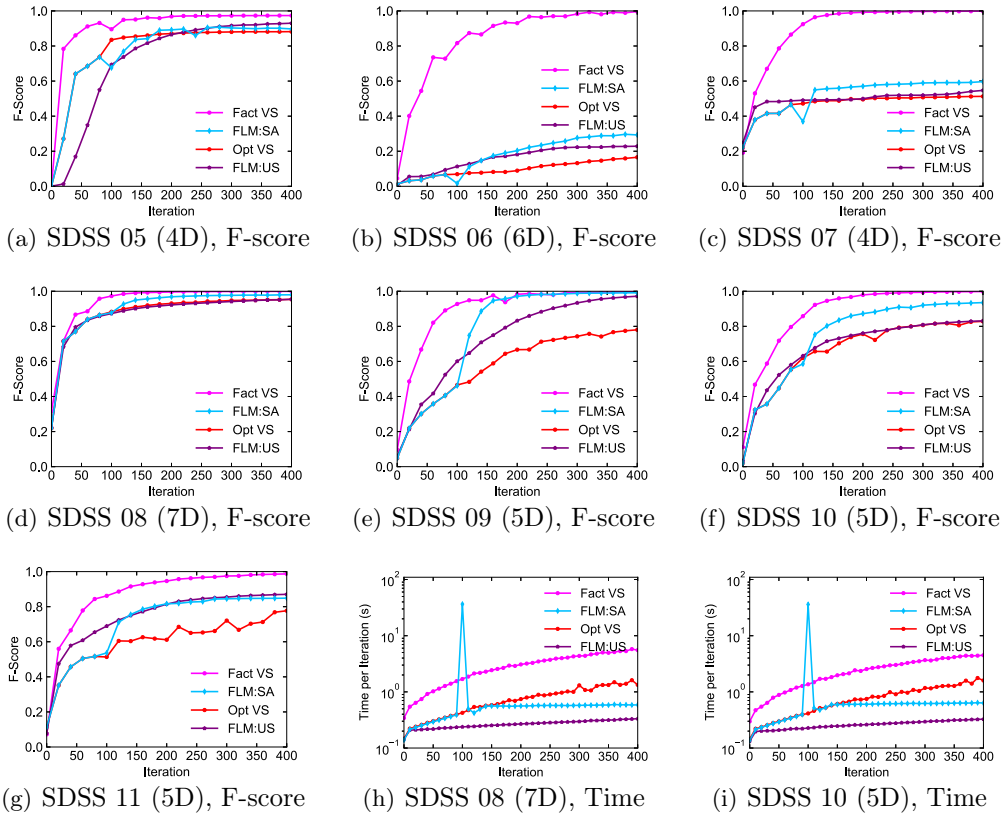
Experimental Setup

As in the previous chapters, the experimental setup is almost identical to the one of Section 4.4, with the few differences being explained below:

Data sets and user interest patterns: We run our experiments under the same data sets and user patterns of the previous section: all Car queries and only high-dimensional SDSS queries (05–11) to compare with factorized algorithms.

Algorithms: We compare the Uncertainty Sampling (FLM:US) strategy of Equation 6.17 and the Swapping Algorithm (FLM:SA) described in Algorithm 6 to the version space algorithms developed in the previous chapters, OptVS and Factorized Version Space. We do not consider QBD [Settles, 2012], Simple Margin [Tong and Koller, 2001], and DSM [Huang et al., 2018] due to their comparatively lower performance to our VS methods.

Hyper-parameter tuning: First, the Uncertainty Sampling strategy trains similarly as Unpenalized FLM, using a Adam optimizer without mini-batches, a constant 0.1 step-size, and 1000 optimization iteration. The Swapping Algorithm uses OptVS as active learner with the same parameters as in Section 4.4. For the swap limit, we use $N_{swap} = 100$ for SDSS queries and $N_{swap} = 20$ for Car queries, given the different rates of convergence in both data sets. As for the FLM parameters, we generally use $S = 10$, $m = 500k$, and $\lambda_1 = \lambda_2 = 10^{-4}$. In all cases, Adam is used for optimization with default parameters and an initial step size of 0.1. The remaining optimization configurations depend on the data set and the phase in question. For the Swapping phase, SDSS queries use the same configurations as in the batch case, and Car queries train with full batches for 5000 iterations. In the Factorized phase, Adam is also trained with full batches for a number of iterations given by the exponential scheduler of Equation 6.19 with parameters $m_1 = 10$, $m_2 = 1000$, and $r = 5$.



(a) SDSS 05 (4D), F-score (b) SDSS 06 (6D), F-score (c) SDSS 07 (4D), F-score

(d) SDSS 08 (7D), F-score (e) SDSS 09 (5D), F-score (f) SDSS 10 (5D), F-score

(g) SDSS 11 (5D), F-score (h) SDSS 08 (7D), Time (i) SDSS 10 (5D), Time

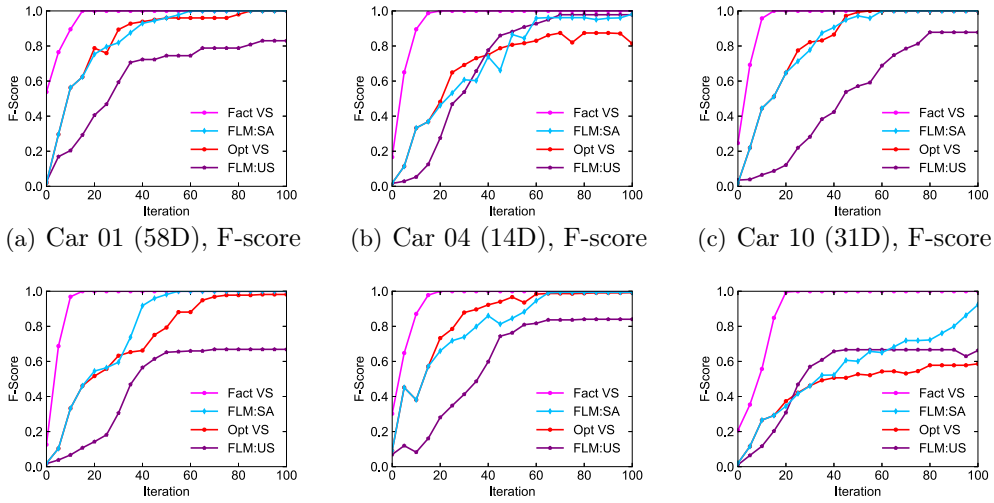
Figure 6.5: Comparison of the Swapping Algorithm with our other active learning techniques in terms of classification accuracy and time per iteration using the SDSS data set. In addition, we display in parenthesis the dimensionality of each query.

Comparison of our FLM-based Techniques

In this first experimental section, we wish to verify how the two FLM-based active learning strategies we propose, Uncertainty Sampling (FLM:US) and the Swapping Algorithm (FLM:SA), compare in practice. Figures 6.5 and 6.6 show a comparison of all algorithms over selected SDSS and Car queries, while Tables 6.4 and 6.5 display a complete set of results.

To simplify our comparison, we split the analysis across two main scenarios:

- *Early iterations*: We first consider the labeling iterations before the swap point, where FLM:SA reduces to OptVS. In this case, we observe that FLM:SA outperforms FLM:US over all Car queries and achieve similar or better performance over most SDSS patterns. For example, Figure 6.5(a) displays the results for SDSS 05, where FLM:SA scores up to 40 F-score



(a) Car 01 (58D), F-score (b) Car 04 (14D), F-score (c) Car 10 (31D), F-score

(d) Car 11 (26D), F-score (e) Car 15 (51D), F-score (f) Car 17 (49D), F-score

Figure 6.6: Comparison of the Swapping Algorithm with our other active learning techniques in terms of classification accuracy using the Car data set. In addition, we display in parenthesis the dimensionality of each query after one-hot encoding.

points higher than FLM:US during the initial 100 labeled examples. This difference in performance is expected since VS-based techniques are generally more efficient than uncertainty sampling methods. More precisely, US only considers the output of a single classifier for selecting the next point to label, while OptVS can leverage the extra information from the hypothesis set by *pooling* the predictions of several classifiers [Settles, 2012].

- *Later iterations*: Next, we consider the behavior after the swap iteration, where FLM:SA switches to a penalized US strategy. First, we note that no drop in performance is observed in the initial iterations after the classifier swap, showing the effectiveness of our incremental training procedure for slowly adapting between the previous and current AL strategies. Additionally, we observe that FLM:SA achieves similar or better performance than FLM:US in all cases. For example, Figures 6.6(a) and 6.6(c) display the results for Car 01 and 10. For both queries, our Swapping Algorithm reaches perfect accuracy after 60 labeling iterations, while FLM:US is still below 70% F-score at this point and remains unable to match our performance even after 100 labeled examples.

Another interesting point to notice is that the Swapping Algorithm can surpass FLM:US even when it swaps at a lower accuracy point, as seen in Figures 6.5(e) and 6.5(g). This behavior suggests that the FLM:US strategy can benefit from using penalty terms and improve its convergence speed. . However,

Query	FLM:SA			Opt VS			FLM:US		
	100	200	400	100	200	400	100	200	400
05	74	**90**	92	**84**	87	88	69	87	**93**
06	3	**23**	**29**	7	9	17	**11**	18	23
07	37	**57**	**60**	47	50	51	**49**	50	55
08	**88**	**96**	**98**	**88**	93	95	87	92	95
09	46	**96**	**99**	47	67	78	**60**	83	97
10	58	**89**	**94**	62	76	83	**63**	76	83
11	53	**83**	85	51	61	78	**69**	81	**87**

Table 6.4: F-score (in %) comparison at iterations 100, 200, and 400 of the Swapping Algorithm with other active learning techniques using the SDSS data set. Bold values represent the observed maximum across all algorithms per query at the given iteration. Fact VS outperforms all algorithms at all times.

using penalty terms from the exploration's beginning can be detrimental to performance since the lack of labeled data may lead to overly simplistic models with too many discarded features and subspaces. Our FLM:SA avoids this problem by first fitting a moderately accurate FLM in the Swapping phase and then incrementally training a penalized FLM over the following iterations. Overall, we conclude that an adaptive approach that gradually increases the penalty may be beneficial for FLM-based strategies, but any further developments will be left to future work.

In summary, our FLM:SA strategy was shown to outperform FLM:US in the vast majority of user patterns considered and will become our default FLM-based algorithm in what follows.

Comparing with our Version Space Algorithms

Next, we observe how our automatically factorized algorithm, FLM:SA, compares with the version space algorithms developed in the previous chapters, OptVS and Fact VS. We include OptVS as a baseline for non-factorized active learners since we observed it outperform related techniques in Section 4.4. Fact VS, on the other hand, is treated as an upper bound for our FLM-based strategy since it has access to the user's true factorization, while FLM:SA has to learn one from the user-labeled data.

First, we compare our new technique with OptVS across each phase of the exploration process. First, there is no performance difference in the *Active Learning phase* since FLM:SA reduces to OptVS over these early iterations. At the *Swapping phase*, we observe a slight drop in performance when compared to OptVS, amounting to 2.4 F-score points on average. This small difference indicates that

Query	FLM:SA			Opt VS			FLM:US		
	20	50	100	20	50	100	20	50	100
01	75	**96**	**100**	**79**	**96**	**100**	41	74	83
02	89	**97**	**100**	**91**	95	97	48	96	**100**
03	54	**99**	**100**	**67**	93	**100**	62	**99**	**100**
04	46	87	**98**	**48**	81	81	28	**88**	**98**
05	99	**100**	**100**	**100**	**100**	**100**	86	**100**	**100**
06	**100**	**100**	**100**	**100**	**100**	**100**	78	99	**100**
07	**81**	91	97	77	**93**	98	40	**100**	**100**
08	**100**	**100**	**100**	**100**	**100**	**100**	86	**100**	**100**
09	97	**100**	**100**	**98**	**100**	**100**	86	**100**	**100**
10	**65**	97	**100**	**65**	**99**	**100**	12	57	88
11	**54**	**98**	**100**	52	79	98	14	65	67
12	71	**100**	**100**	**80**	99	99	55	94	94
13	69	**99**	**100**	**78**	**99**	**100**	23	**99**	**100**
14	93	99	99	**94**	**100**	**100**	46	96	**100**
15	66	85	**99**	**73**	**97**	**99**	28	76	84
16	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
17	34	60	**92**	**37**	53	59	31	**67**	66
18	**99**	**100**	**100**	91	**100**	**100**	70	91	91

Table 6.5: F-score (in %) comparison at iterations 20, 50, and 100 of the Swapping Algorithm with other active learning techniques using the Car data set. Bold values represent the observed maximum across all algorithms per query at the given iteration. Fact VS outperforms all algorithms at all times.

the FLM model built at this step can accurately mimic OptVS's predictions, thus providing a good starting point for the uncertainty sampling phase that follows. Finally, in the *Factorization phase*, we observe that our FLM:SA achieves similar or better performance than OptVS in all cases; for example, Figure 6.6(d) displays the results for Car 11, where the Swapping Algorithm reaches 100% accuracy after 55 iterations while OptVS is still at 80% accuracy.

Next, we evaluate how our factorization-aware algorithm compares with the Swapping Algorithm. In terms of classification accuracy, Fact VS outperforms all other methods at every single iteration for all queries we considered. This behavior is expected since this model leverages more information from the user than other strategies: not only the factorization structure is known from the beginning, but the user also provides the *subspace labels* at every iteration, which effectively enables a decomposition of the learning problem into independent, simpler subproblems. This behavior is also compatible with the experimental results of Section 5.7, in which pure active learning strategies were also unable to match the performance of explicitly factorized algorithms. However, we also note that

FLM:SA was observed to match Fact VS's performance in some scenarios. For example, Figure 6.5(e) depicts the case of SDSS 09, where both the FLM:SA and Fact VS managed to achieve nearly perfect accuracy after 150 labeled iterations, while the other techniques were still at 70% F-score or below.

In summary, FLM:SA provides an edge over non-factorized exploration strategies while approximating the performance of explicitly factorized methods without requiring any extra information from the user.

Time Measurements

To conclude the experimental evaluation, we observe how the Swapping Algorithm performs in terms of time per iteration. Let us consider Figures 6.5(h) and 6.5(i) representing the time measurements for the two most expensive SDSS queries, 08 and 10. At every iteration except at the swapping point, we can see that our FLM:SA either performs identically as OptVS or close to FLM:US, taking less than 1 second per iteration at all times and thus running under interactive performance.

We also note that the Swapping phase can present a high time cost, taking almost 40 seconds to finish in the above cases. We expect this higher cost because of the comparatively larger sample of data points used to train the FLM classifier at this phase: 200k points for SDSS and around 5k points for Car. However, in practice, we can avoid this problem by delaying the algorithm swap for a few labeling iterations. More precisely, once we reach the swap iteration, we can run the costly FLM training procedure in a background thread and delay the classifier swap until the training completes. Meanwhile, the user could continue to label the new data points chosen by OptVS for a few extra iterations.

Summary of Experimental Results

From the previous experiments, we can draw the following conclusions regarding our novel Swapping Algorithm:

- *Comparison to non-factorized strategies*: FLM:SA was observed to outperform another FLM-based exploration strategy, FLM:US, as well as a baseline non-factorized active learner, OptVS. For example, in the particular case Car 11, the Swapping Algorithm reaches 100% accuracy after 55 iterations, while OptVS and FLM:US are still at 80% and 60% F-score.
- *Comparison to explicitly factorized strategies*: Additionally, the Swapping Algorithm was shown to approximate the performance of Fact VS, a less realistic upper bound that relies on extra user information. In particular, we still observed a couple of cases where both methods achieve similar performances; for example, over SDSS 09 both FLM:SA and Fact VS managed to

achieve nearly perfect accuracy after 150 labeled iterations, while the other techniques were still at a 70% F-score or below.

- *Time measurements*: FLM:SA was shown to run under 1 second per iteration for all queries and almost every iteration, thus running under interactive performance.

6.5 Chapter Summary

In this chapter, we have developed the following contributions:

- *Learning Factorized Classifiers*: Inspired by the user interest patterns from our user studies, we devised a novel factorized classification algorithm, called the Factorized Linear Model (FLM), that mimics the human decision-making process under factorization. In particular, the FLM decomposes the user interest as a combination of low-dimensional convex objects that, together, give rise to the final prediction. Additionally, we have also introduced a few extensions of this algorithm, including an extension to kernel classifiers and optimizations for categorical variables.

- *A Swapping Algorithm for Efficient Data Exploration*: In the active learning scenario, we have also introduced a new algorithm called the Swapping Algorithm, a hybrid exploration strategy between our OptVS and FLM-based Uncertainty Sampling. This algorithm initially runs OptVS for a certain number of iterations, enjoying its fast initial convergence speed, and then swaps to an FLM-based strategy which can reach higher classification accuracy due to its factorized classifier. Additionally, we have devised a couple of techniques to slowly transition between exploration strategies, thus avoiding any drop in performance.

- *Evaluation (batch learning)*: In the batch case, we observed that both our non-interpretable linear (Unpenalized FLM) and kernel (FKM) techniques outperform SVM in at least 80% of cases. We also noticed a similar behavior for our interpretable methods, with our Penalized FLM defeating another human-interpretable algorithm, VIPR [Fiterau and Dubrawski, 2012], in 20 out of all 25 user patterns. In terms of running time, our proposed methods take at most a few minutes to train over a large data set of nearly one million points.

- *Evaluation (active learning)*: In the active learning case, we observed that the Swapping Algorithm outperforms OptVS, a baseline non-factorized active learner, while approximating the explicitly factorized Fact VS, a less realistic upper bound which relies on extra user information. In the particular case of a complex SDSS pattern, both the Swapping Algorithm and

Fact VS achieved nearly perfect accuracy after 150 labeling iterations, while other techniques were still at 70% F-score and below.

Chapter 7

Conclusions and Future Work

7.1 Thesis Summary

To overcome the slow convergence problem of active learning for human-in-the-loop model development over large datasets, we presented five major contributions:

A New Theoretical Framework for Version Space Algorithms over Kernel Classifiers. We proposed a new theoretical framework that allows for an efficient implementation of the Generalized Binary Search strategy over kernel classifiers. Compared to previous work, our technique offers several advantages from both theoretical and practical standpoints, including (*i*) strong theoretical bounds on the number of iterations until convergence, (*ii*) a dimensionality reduction technique that restricts kernel computations to the set of labeled points, (*iii*) a new Cholesky-decomposition-based technique to efficiently estimate the version space reduction of any sample and, finally, (*iv*) generalization error bounds on accuracy and F-score that enable our algorithms to run over a sample from the original large data set with minimal performance loss.

An Optimized Version Space Algorithm. Based on our theoretical results, we developed an optimized version space algorithm called OptVS. Our implementation uses the hit-and-run algorithm for sampling the version space enhanced with several optimizations to reduce the time cost, including (*i*) using the burn-in and thinning methods for extracting all samples from a single hit-and-run chain, (*ii*) a new technique for caching the rounding ellipsoid between labeling iterations for efficient computation, and (*iii*) an optimized rounding algorithm that is numerically stable and avoids the expensive diagonalization of the scaling matrix. Our evaluation results show that OptVS outperforms state-of-the-art version space algorithms in the vast majority of user patterns considered while still running under 1 second per iteration after hundreds of labeled examples.

A Factorized Version Space Algorithm. 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. In particular, we have developed three new factorized selection strategies, Greedy, Squared-Loss, and Product-Loss, each offering a different trade-off between convergence speed and theoretical guarantees on performance. Our methods also support an optimization for categorical subspaces which significantly improves the running time while having little to no impact on performance. In practice, our factorized techniques outperform VS algorithms and another factorization-aware algorithm, DSM, often by a wide margin while maintaining interactive speed. For example, for a complex user interest pattern tested, our algorithm achieves an F-score of over 80% after 100 labeling iterations while DSM is still at 40% and all other VS algorithms are at 10% or lower.

A Learning Algorithm for Factorized Classifiers. Next, we developed a new human-inspired classification algorithm, called the Factorized Linear Model, that breaks the prediction process as a collection of low-dimensional classification tasks, mimicking the decision-making process of real users. In particular, the FLM is able to decompose the user interest as a combination of convex objects in low-dimensional spaces, which results in a model that is accurate, efficient, and interpretable by humans. Additionally, we proposed a few extensions that enhance the applicability of our technique, including an extension to kernel classifiers and optimizations for categorical attributes. Our evaluation shows that the FLM classifier achieves comparable or better performance than SVM and another interpretable model, VIPR, over 80% of all considered user interest patterns while managing to train over nearly one million data points in a couple of minutes.

A Swapping Algorithm for Efficient Data Exploration. Finally, we also proposed a new automatically factorized active learning strategy called the Swapping Algorithm. This technique initially employs an efficient active learner like OptVS to escape the slow convergence of initial iterations and then swaps to an FLM-based Uncertainty Sampling strategy to take advantage of its higher classification accuracy. In doing so, the Swapping Algorithm manages to outperform OptVS, a baseline non-factorized active learner, while approximating the explicitly factorized Fact VS, a less realistic upper bound which relies on extra user information. For example, in the particular case of a complex SDSS pattern, both the Swapping Algorithm and Fact VS achieved nearly perfect accuracy after 150 labeling iterations, while other techniques were still at 70% F-score and below.

7.2 Future Work

In the future, there are several lines of work that could be pursued based on our contributions:

Noisy Labeling. In real-world exploration tasks, the human user is likely to

make annotation mistakes throughout the exploration process, especially when the point to label is close to the decision boundary between the "interesting" and "not interesting" sets. These labeling mistakes can be very detrimental to the performance of active learning algorithms [Settles, 2012], and especially so for version space techniques where it may result in an empty version space. To counter this problem, a few directions can be explored. First, one could try to identify which labeled points are most likely to be noisy and remove them from consideration. Another approach could be to extend version space techniques to handle mistakes; for example, the version space definition could be modified to support "up to K" labeling errors, where K is an upper bound to the noise rate. However, how to devise efficient sampling techniques for such cases is still an open problem.

Adding a Factorization in the Middle of the Exploration. One disadvantage of factorized active learners is that a factorization structure must be provided by the user at the beginning of the exploration process. However, it is more natural to assume that a factorization may only be available *after* the user annotates several labeled points and a better understanding of his own decision-making process has been built. One problem with this approach is that the initial labeled points do not have their subspace labels, and there might be too many such points to ask the user for relabeling. Thus, one possible line of future work would be to develop an algorithm for inferring the missing subspace labels. In particular, one could think of enhancing the selection process in subsequent iterations by giving preference to data points which help us uncover the missing labels. Another interesting approach could be to model the missing subspace labels as latent variables and then adopt an Expectation-Maximization optimization procedure to simultaneously infer these missing values and fit an accurate factorized classification model.

Extending the Factorization Framework. In our work, we have only considered how to extend the version space framework to support a factorization structure. In this regard, one could also explore how other classes of active learning algorithms, such as Uncertainty Sampling or Expected Error Reduction methods, could also be extended to support the extra factorization information.

Data Programming Integration. Furthermore, another interesting line of future work is to 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. In particular, it would be advantageous to combine our fast convergence in early iterations of human labeling with the auto-labeling functionality of Data Programming to scale to large datasets.

Support for Non-Convex Patterns. One major limitation of our FLM classifier is it cannot accurately learn non-convex user patterns. To lift this limitation, a few lines of work can be explored. First, one natural approach is to find a decomposition of the user interest pattern as the union of convex

objects and train one FLM over each piece. Another option is to take inspiration from VIPR [Fiterau and Dubrawski, 2012] and learn a partitioning of the data into easy-to-classify subsets. We expect that this approach will probably require drastic modifications to the loss function and optimization procedure.

Bibliography

F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *International Conference on Machine Learning (ICML)*, page 33–40, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102356. URL https://doi.org/10.1145/1102351.1102356.

S. H. Bach, B. He, A. Ratner, and C. Ré. Learning the structure of generative models without labeled data. In *International Conference on Machine Learning*, volume 70, pages 273–282, 2017.

C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, Dec. 1996. ISSN 0098-3500. doi: 10.1145/235815.235821.

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

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

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004. ISBN 0521833787.

L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324.

R. Bresson, J. Cohen, E. Hüllermeier, C. Labreuche, and M. Sebag. Neural representation and learning of hierarchical 2-additive choquet integrals. In C. Bessiere, editor, *International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1984–1991, 7 2020. doi: 10.24963/ijcai.2020/275.

L. Bull, K. Worden, G. Manson, and N. 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.

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

J. Chiquet, Y. Grandvalet, and G. Rigaill. On coding effects in regularized categorical regression. *Statistical Modelling*, 16(3):228–237, June 2016. doi: 10.1177/1471082X16644998.

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

D. De Martino, M. Mori, and V. 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.

K. Dimitriadou, O. Papaemmanouil, and Y. 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.

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

J. Figueira, S. Greco, and M. Ehrgott. Multiple criteria decision analysis, state of the art surveys. *Multiple Criteria Decision Analysis: State of the Art Surveys*, 78, 01 2005. doi: 10.1007/b100605.

M. Fiterau and A. Dubrawski. Projection retrieval for classification. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

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

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

D. Golovin and A. 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.

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

M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research (JMLR)*, 12:2211–2268, July 2011. ISSN 1532-4435.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

D. E. Grierson. Pareto multi-criteria decision making. *Advanced Engineering Informatics*, 22:371–384, 07 2008. doi: 10.1016/j.aei.2008.03.001.

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

R. L. Harms and A. 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.

T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC, 2015. ISBN 1498712169.

D. Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(88)90002-1.

H. Hirsh. Incremental version-space merging. In *International Conference in Machine Learning (ICML)*, 1990.

R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90, 1993.

T. Hong and S. Tseng. A generalized version space learning algorithm for noisy and uncertain data. *IEEE Trans. Knowl. Data Eng.*, 9:336–340, 1997.

E. Huang, L. Peng, L. D. Palma, A. Abdelkafi, A. Liu, and Y. Diao. Optimization for active learning-based interactive database exploration. *Proceedings of the VLDB Endowment*, 12(1):71–84, 2018.

E. Huang, L. D. Palma, L. Cetinsoy, Y. Diao, and A. Liu. AIDEme: An active learning based system for interactive exploration of large datasets. NeurIPS Demonstration Track, Dec. 2019.

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

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

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations (ICLR)*, 2015.

A. Krishnamurthy, A. Agarwal, T.-K. Huang, H. Daumé, and J. Langford. Active learning for cost-sensitive classification. In *International Conference on Machine Learning*, volume 70, pages 1915–1924, 2017.

B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3), Sep 2015. ISSN 1932-6157. doi: 10.1214/15-aoas848.

W. A. Link and M. 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.

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

L. 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.

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

G. 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.

T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, page 305–310. Morgan Kaufmann Publishers Inc., 1977.

T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(82)90040-6.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X.

C. Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

R. M. Monarch. *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Manning Publications, 2021. ISBN 978-1617296741.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020.

P. Nemery and A. Ishizaka. *Multi-Criteria Decision Analysis: Methods and Software*. 05 2013. ISBN 978-1-119-97407-9. doi: 10.1002/9781118644898.ch8.

D. Pohl, A. Bouchachia, and H. 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.

S. Qiu and T. Lane. A framework for multiple kernel support vector regression and its applications to sirna efficacy prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):190–199, Apr. 2009. ISSN 1545-5963. doi: 10.1109/TCBB.2008.139.

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

D. Reinsel, J. Gantz, and J. Rydning. The Digitization of the World: From Edge to Core. Technical report, International Data Corporation (IDC), November 2018. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972. ISSN 0097-3165. doi: 10.1016/0097-3165(72)90019-2.

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

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

A. J. Smola and B. Schökopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning (ICML)*, page 911–918, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607072.

J. Song, H. Wang, Y. Gao, and B. 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.

C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007.

A. K. M. K. A. Talukder and K. Deb. Paletteviz: A visualization method for functional understanding of high-dimensional pareto-optimal data-sets to aid multi-criteria decision making. *IEEE Computational Intelligence Magazine*, 15 (2):36–48, 2020. doi: 10.1109/MCI.2020.2976184.

H. Tanabe, T. B. Ho, C. H. Nguyen, and S. Kawasaki. Simple but effective methods for combining kernels in computational biology. In *IEEE International Conference on Research, Innovation and Vision for the Future in Computing & Communication Technologies*, pages 71–78. IEEE, 2008. doi: 10.1109/RIVF.2008.4586335.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

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

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

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

C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001.

L. Yang, Y. Zhang, J. Chen, S. Zhang, and D. 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.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. doi: 10.1111/j.1467-9868.2005.00532.x.

Appendix A

Proof of Dimensionality Reduction Lemma (Lemma 3.8)

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

Lemma A.1. *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*$(a_1, \ldots, a_t, a^*)$, *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 for the subspace $\mathbb{R}^k \times \{0\}^{N-k}$, then it must also hold 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 $\tilde{a}_i = (a_i^1, \ldots, a_i^k)$. 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 us define the random vector $\omega = (U_1, \ldots, U_k, V_1, \ldots, V_{N-k}) \in \mathbb{R}^N$. This vector is distributed according to

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

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

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
- $S_j = \mathbb{R} \times span(L_1, \ldots, L_t, L_j)$ clearly contains $span(a_1, \ldots, a_t, a^*)$

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 $\mathcal{V}_\beta \cap S_j$.

Appendix B

Hit-and-Run Implementation

Let $W \subset \mathbb{R}^n$ be a non-empty bounded convex set. Hit-and-run is a randomized algorithm for generating a uniformly random sample from any convex body. More precisely, this algorithm generates a Markov Chain inside W that converges to the uniform distribution. This Markov Chain can start at any interior point $w_0 \in W$ and then proceeds by iteratively performing the following two steps:

1. *Hit step*: Generate a random line L passing through w_t and compute its intersection with W
2. *Run step*: Set w_{t+1} as a random point on the line segment $L \cap W$.

Now, we focus on how to implement this algorithm for convex bodies of the form $W = P \cap B_n$, where P is a polytope $P = \{w \in \mathbb{R}^n : Aw \leq 0\}$ and B_n is unit ball.

The first step is to generate a random line going through w_t. This problem is equivalent to sampling a vector D uniformly over the unit sphere, which can be easily done through Marsaglia's Algorithm [Marsaglia, 2007]: simply sample $D \sim \mathcal{N}(0, I_d)$ and normalize the resulting vector. Once this step is completed, the random line is given by $L = \{w_t + sD,\, s \in \mathbb{R}\}$.

Next, the Hit step requires computing the intersection between L and W, which can be done via simple calculations. First, the intersection with the polytope P is given by

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

Next, we consider the intersection with the unit ball B_n. By keeping in mind

Algorithm 7 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 $\{w_1, w_2, \ldots, w_N\}$

$samples \leftarrow \{\}$
for t from 0 to $N-1$ **do**
  $D \leftarrow T^{-1}\mathcal{N}(0, I_n)$
  $D \leftarrow D/\|D\|$
  $L_{pol}, U_{pol} \leftarrow \max_{a_i^T D<0} -\frac{a_i^T w_t}{a_i^T D}, \min_{a_i^T D>0} -\frac{a_i^T w_t}{a_i^T D}$
  $\Delta \leftarrow (w_t^T D)^2 - \|w_t\|^2 + 1$
  $L_{ball}, U_{ball} \leftarrow -w_t^T D - \sqrt{\Delta}, -w_t^T D + \sqrt{\Delta}$
  $L, U \leftarrow \max(L_{pol}, L_{ball}), \min(U_{pol}, U_{ball})$
  $s \leftarrow Unif([L, U])$
  $w_{t+1} \leftarrow w_t + sD$
  $samples \leftarrow samples \cup \{w_{t+1}\}$
end for
return *samples*

that $\|D\| = 1$, we have that

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

where $\Delta = (w_t^T D)^2 - \|w_t\|^2 + 1$. We note that Δ is always positive since $w_t \in B_n$.

Finally, we can conclude that the intersection $L \cap W$ is simply given by $\{w_t + sD,\ s \in [s_{min}, s_{max}]\}$, where $[s_{min}, s_{max}]$ is the intersection of the above two intervals computed. In particular, the Run step can now be easily implemented by sampling S uniformly over $[s_{min}, s_{max}]$ and then setting $w_{t+1} = w_t + SD$.

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 7 for a pseudo-code of this entire process.

Appendix C

Rounding Implementation

This appendix gives 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 $\mathcal{E} \cap H$, 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 8.

In the particular case of our rounding algorithm, only the third case above has to be considered. 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 8.

Another point to note is that Algorithm 8 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 4.2, T is chosen as any

Algorithm 8 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 $\mathcal{E} \cap H$

$\hat{a} \leftarrow L^T a$
$\eta \leftarrow \sqrt{\sum_i d_i \hat{a}_i^2}$
$\alpha \leftarrow (a^T z - b)/\eta$
$p \leftarrow \frac{1}{\eta} D\hat{a}$
$\tau \leftarrow (1 - n\alpha)/(n+1)$
$z' \leftarrow z + \tau Lp$
$\delta \leftarrow n^2(1-\alpha^2)/(n^2-1)$
$\sigma \leftarrow 2\tau/(1-\alpha)$
$t_{n+1} \leftarrow \frac{n-1}{n+1}\frac{1-\alpha}{1+\alpha}$
for $j = n, n-1, \ldots, 1$ **do**
  $t_j \leftarrow t_{j+1} + \sigma p_j^2/d_j$
  $d'_j \leftarrow \delta d_j t_{j+1}/t_j$
  $\xi_j \leftarrow -\sigma p_j/(d_j t_{j+1})$
  $v_j^{(j)} \leftarrow p_j$
  for $r = j+1, \ldots, n$ **do**
    $L'_{rj} \leftarrow L_{rj} + \beta_j v_r^{(j+1)}$
    $v_r^{(j)} \leftarrow v_r^{(j+1)} + p_j L_{rj}$
  end for
end for
return $\mathcal{E}(z', P' = L'D'L'^T)$

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

In this appendix, we provide an overview of all user interest patterns used in our experiments, including their most relevant properties, which can be found on Table D.1. In addition, we note that subspaces are obtained by splitting on **AND**s.

The 11 SDSS queries and 18 Car queries are:

SDSS 01: $662.5 < rowc < 702.5$ AND $991.5 < colc < 1053.5$
SDSS 02: $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 29^2$
SDSS 03: $190 < ra < 200$ AND $53 < dec < 57$
SDSS 04: $rowv^2 + colv^2 > 0.5^2$
SDSS 05: $(rowc-682.5)^2+(colc-1022.5)^2 < 90^2$ **AND** ($180 < ra < 210$ AND $50 < dec < 60$)
SDSS 06: $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 280^2$ **AND** ($150 < ra < 240$ AND $40 < dec < 70$) **AND** $rowv^2 + colv^2 > 0.2^2$
SDSS 07: $x_1 > (1.35+0.25*x_2)$ **AND** $x_3+2.5*\log(2*3.1415*petror50\_r^2) < 23.3$
SDSS 08: $(dered\_r-dered\_i) < 2$ **AND** $17.5 \leq cmodelmag\_i-extinction\_i \leq 19.9$ **AND** $dered\_r-dered\_i-(dered\_g-dered\_r)/8 > 0.55$ **AND** $fiber2mag\_i < 21.7$ **AND** $devrad\_i < 20.$ **AND** $dered\_i < 19.86+1.6*((dered\_r-dered\_i)-(dered\_g - dered\_r)/8. - 0.80)$
SDSS 09: $u - g < 0.4$ **AND** $g - r < 0.7$ **AND** $r - i > 0.4$ **AND** $i - z > 0.4$
SDSS 10: $g \leq 22$ **AND** $-0.27 \leq u - g \leq 0.71$ **AND** $-0.24 \leq g - r \leq 0.35$ **AND** $-0.24 \leq r - i \leq 0.57$ **AND** $-0.35 \leq i - z \leq 0.70$
SDSS 11: ($u - g > 2.0$ OR $u > 22.3$) **AND** $0 \leq i \leq 19$ **AND** $g - r > 1$ **AND** $r - i < 0.08 + 0.42 * (g - r - 0.96)$ OR $g - r > 2.26$ **AND** $i - z < 0.25$
Car 01: $class$ = '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')
Car 02: $price\_msrp \leq 22132$ **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** $fuel\_tank\_capacity \geq 65$

Car 03: *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'
Car 04: *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
Car 05: *class* = 'full-size van' **AND** *body_type* = 'van' **AND** *engine_type* = 'gas' **AND** *model* = 'nv cargo' **AND** *price_msrp* $<$ 27000 **AND** *horsepower* $<$ 300
Car 06: *height* $<$ 1.51 **AND** *drivetrain_year* $\geq$ 10 **AND** *class* = 'subcompact car' **AND** (*transmission* $\neq$ '6-speed manual' **AND** *transmission* $\neq$ '7-speed manual')
Car 07: *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'
Car 08: *length* $\geq$ 6 **AND** *body_type* $\neq$ 'sedan' **AND** *drive_type* $\neq$ 'front wheel drive' **AND** *basic_year* $\geq$ 4 **AND** *drivetrain_year* $\geq$ 5 **AND** *price_msrp* $<$ 32000 **AND** *height* $>$ 2.5 **AND** (*fuel_type* $\neq$ 'premium unleaded (required)' AND *fuel_type* $\neq$ 'premium unleaded (recommended)')
Car 09: (*body_type* = 'truck' OR *body_type* = 'van') **AND** *price_msrp* $<$ 30000 **AND** *height* $\geq$ 2.5 **AND** *length* $>$ 6
Car 10: *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
Car 11: (*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%'
Car 12: *price_msrp* $<$ 25000 **AND** *horsepower* $>$ 200 **AND** *year* = 2017 **AND** *length* $>$ 5.5
Car 13: *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
Car 14: *price_msrp* $<$ 13000 **AND** *drive_type* = 'front wheel drive' **AND** *transmission* LIKE '%manual%' **AND** *length* $<$ 4.5 **AND** *horsepower* $<$ 110
Car 15: *basic_year* $\geq$ 4 **AND** *year* $\geq$ 2017 **AND** *height* $\leq$ 1.62 **AND** *price_msrp* $<$ 25000 **AND** *transmission* LIKE '%automatic%' **AND** (*class* NOT LIKE '%pickup%' AND *class* NOT LIKE '%suv%')
Car 16: *price_msrp* $<$ 26000 **AND** *body_type* IN ('van', 'truck') **AND** 2 $<$ *height* $<$ 2.5 **AND** *basic_year* $>$ 3
Car 17: *horsepower* $>$ 150 **AND** *year* = 2017 **AND** *make* IN ('hyundai', 'honda') **AND** *length* $<$ 4.5 **AND** *engine_type* IN ('gas', 'diesel') **AND** *price_msrp* $<$ 20000

Car 18: *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

Query	Selectivity (%)	# Features	# Subspaces	Is Convex
SDSS 01	0.1	2	1	T
SDSS 02	0.1	2	1	T
SDSS 03	0.1	2	1	T
SDSS 04	0.1	2	1	T
SDSS 05	0.01	4	2	T
SDSS 06	0.01	6	3	F
SDSS 07	7.8	4	2	F
SDSS 08	5.5	7	5	T
SDSS 09	1.5	5	4	T
SDSS 10	0.5	5	5	T
SDSS 11	0.1	5	5	F
Car 01	0.32	6 (58)	6	T
Car 02	0.27	8 (37)	8	T
Car 03	0.27	8 (35)	6	T
Car 04	0.25	6 (14)	6	T
Car 05	0.23	6 (418)	6	T
Car 06	0.34	4 (49)	4	T
Car 07	0.82	6 (59)	6	T
Car 08	0.32	8 (26)	8	T
Car 09	0.30	4 (12)	4	T
Car 10	0.36	8 (31)	8	T
Car 11	0.29	6 (26)	6	T
Car 12	0.20	4	4	T
Car 13	0.29	10 (24)	10	T
Car 14	0.23	5 (29)	5	T
Car 15	0.91	6 (51)	6	T
Car 16	0.21	4 (12)	4	T
Car 17	0.20	6 (49)	6	T
Car 18	0.20	6 (17)	6	T

Table D.1: Collection of the most relevant properties of each user interest pattern over SDSS and Car data sets. If a query involves categorical attributes, the "# Features" column also displays in parenthesis the number of features *after* one-hot-encoding.

ECOLE DOCTORALE

Titre : Nouveaux algorithmes et optimisations pour le développement de modèles interactives

Mots clés : Développement de Modèles Interactives, Big Data, Active Learning, Methodes a noyau, Version Space, Factorization

Résumé : Dans cette thèse, nous proposons un cadre ”human-in-the-loop” pour le développement efficace de modèles sur de grands ensembles de données. Dans ce cadre, nous visons à appliquer des algorithmes d'apprentissage actif pour sélectionner une petite séquence d'instances de données que l'utilisateur doit étiqueter et dériver un modèle précis, tout en offrant une performance interactive en présentant l'instance de données suivante pour examen. Cependant, les techniques d'apprentissage actif existantes ne parviennent souvent pas à fournir des performances satisfaisantes lorsqu'elles sont construites sur de grands ensembles de données. Non seulement ces modèles nécessitent souvent des centaines d'instances de données étiquetées pour atteindre un niveau de précision élevé, mais la récupération de l'instance suivante à étiqueter peut prendre beaucoup de temps, ce qui la rend incompatible avec la nature interactive du processus d'exploration humain. Pour résoudre ces problèmes, nous proposons les contributions suivantes :

1) Un nouveau cadre théorique qui permet une mise en œuvre efficace de la stratégie de recherche binaire généralisée sur les classificateurs à noyau. Par rapport aux travaux précédents, notre cadre offre à la fois de solides garanties théoriques sur les performances et une mise en œuvre efficace en temps et en espace.

2) Un algorithme VS optimisé appelé OptVS qui utilise l'algorithme hit-and-run pour échantillonner l'espace des versions. Nous développons également une série d'optimisations de l'échantillonnage pour améliorer à la fois la qualité de l'échantillon et le temps d'exécution. Dans la pratique, nous observons qu'OptVS atteint des performances similaires ou supérieures à celles des algorithmes d'espace de version de l'état de l'art tout en fonctionnant en permanence en dessous d'une seconde par itération.

3) Un nouvel algorithme qui exploite la structure de factorisation fournie par l'utilisateur pour créer des sous-espaces et factoriser l'espace de version en conséquence pour effectuer un apprentissage actif dans les sous-espaces. Nous fournissons également des résultats théoriques sur l'optimalité de notre algorithme VS factorisé et des optimisations pour traiter les variables catégorielles. Nos résultats d'évaluation montrent que, pour tous les modèles d'utilisateurs considérés, notre algorithme VS factorisé surpasse les apprenants actifs non factorisés ainsi que DSM, un autre algorithme prenant en compte la factorisation, souvent par une large marge tout en maintenant la vitesse interactive.

4) En suivant le raisonnement intuitif derrière le processus de prise de décision de l'utilisateur, nous développons un nouvel algorithme de classification inspiré par l'homme, appelé le modèle linéaire factorisé (FLM), qui décompose l'intérêt de l'utilisateur comme une combinaison d'objets convexes de faible dimension, résultant en un classificateur précis, efficace et interprétable. En pratique, nous observons que le classificateur FLM atteint des performances comparables ou supérieures à celles du SVM et d'un autre modèle interprétable, VIPR, pour la majorité des modèles d'intérêt de l'utilisateur, tout en prenant seulement quelques minutes pour s'entraîner sur un grand ensemble de données de près d'un million de points.

5) Une nouvelle stratégie d'apprentissage actif factorisé automatiquement appelée l'algorithme de permutation. Cette technique utilise initialement OptVS pour échapper à la convergence lente des itérations initiales, puis passe à une stratégie basée sur FLM pour profiter de sa précision de classification supérieure. Notre évaluation montre que l'algorithme de permutation atteint des performances similaires ou supérieures à celles des apprenants actifs non factorisés tout en s'approchant des méthodes explicitement factorisées.

Title : New Algorithms and Optimizations for Human-in-the-Loop Model Development

Keywords : Interactive Model Development, Big Data, Active Learning, Kernel methods, Version Space, Factorization

Abstract : In this thesis, we propose a human-in-the-loop framework for efficient model development over large data sets. In this framework, we aim to apply active learning algorithms to select a small sequence of data instances for the user to label and derive an accurate model while, at the same time, offering interactive performance in presenting the next data instance for reviewing. However, existing active learning techniques often fail to provide satisfactory performance when built over large data sets. Not only such models often require hundreds of labeled data instances to reach high accuracy, but 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 propose the following contributions:

1) A new theoretical framework that allows for an efficient implementation of the Generalized Binary Search strategy over kernel classifiers. Compared to previous work, our framework offers both strong theoretical guarantees on performance and efficient implementation in time and space.

2) An optimized VS algorithm called OptVS that uses the hit-and-run algorithm for sampling the version space. We also develop a range of sampling optimizations to improve both sample quality and running time. In practice, we observe that OptVS achieves similar or better performance than state-of-the-art version space algorithms while running under 1 second per iteration at all times.

3) 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. We also provide theoretical results on the optimality of our factorized VS algorithm and optimizations for dealing with categorical variables. Our evaluation results show that, for all user patterns considered, our factorized VS algorithm outperforms non-factorized active learners as well as DSM, another factorization-aware algorithm, often by a wide margin while maintaining interactive speed.

4) Following the intuitive reasoning behind the user decision-making process, we develop a new human-inspired classification algorithm, called the Factorized Linear Model (FLM), that decomposes the user interest as a combination of low-dimensional convex objects, resulting in an accurate, efficient, and interpretable classifier. In practice, we observe that the FLM classifier achieves comparable or better performance than SVM and another interpretable model, VIPR, over the majority of user interest patterns while taking only a few minutes to train over a large data set of nearly one million points.

5) A novel automatically factorized active learning strategy called the Swapping Algorithm. This technique initially employs OptVS to escape the slow convergence of initial iterations and then swaps to an FLM-based strategy to take advantage of its higher classification accuracy. Our evaluation shows that the Swapping Algorithm achieves similar or better performance than non-factorized active learners while approximating the explicitly factorized methods.

Institut Polytechnique de Paris
91120 Palaiseau, France