Thèse de doctorat de
l'Université Paris VI — Pierre et Marie Curie

Spécialité : Informatique

présentée par

Antoine Bordes

pour obtenir le Grade de Docteur en Sciences
de l'Université Paris VI — Pierre et Marie Curie

New Algorithms for Large-Scale Support Vector Machines

Nouveaux Algorithmes pour l'Apprentissage de Machines à Vecteurs Supports sur de Grandes Masses de Données

soutenue publiquement le 9 février 2010
devant le jury composé de

Jacques Blanc-Talon	Responsable scientifique de l'ingénierie de l'information à la DGA	*Examinateur*
Léon Bottou	Distinguished Senior Researcher à NEC Labs of America	*Examinateur*
Stéphane Canu	Professeur à l'INSA de Rouen	*Rapporteur*
Matthieu Cord	Professeur à l'Université Pierre et Marie Curie	*Président du Jury*
Patrick Gallinari	Professeur à l'Université Pierre et Marie Curie	*Directeur de Thèse*
Bernhard Schölkopf	Professeur au Max Planck Institute for Biological Cybernetics	*Examinateur*
John Shawe-Taylor	Professeur à l'University College London	*Rapporteur*

Prediction is very difficult, especially if it's about the future, *Niels Bohr*

Hé oui, hé oui, l'école est finie, *Sheila*

Acknowledgments

My first thanks are for Léon Bottou who kindly and patiently made me discover and like machine learning research during my first internship at NEC Labs in 2004. His endless knowledge and pertinent intuitions have continuously guided and inspired my thesis work (and still do).

My deepest gratitude goes to my PhD advisor Patrick Gallinari who welcomed me at LIP6 in 2006 for my master's thesis and has always supported me since then, allowing me to enjoy his precious advices and research skills within great working facilities.

I am truthfully grateful to Stéphane Canu and John Shawe-Taylor who accepted the heavy duty of reviewing this dissertation and to Jacques Blanc-Talon, Matthieu Cord and Bernhard Schölkopf for agreeing to be part of the defense jury. I also thank the French *Délégation pour l'Armement* (DGA) for its financial support throughout my thesis.

Most work of this thesis has been developed with excellent collaborators. Apart form Léon and Patrick, I want to acknowledge Jason Weston and Ronan Collobert from NEC Labs who are now far more than co-workers as well as Nicolas Usunier from LIP6 who helped me so much towards the end of this thesis (Nicolas, you got a special thank for actually reading it out). And thank you guys for still supporting, growing and believing in the bAbI project with me!

A big round of applause now goes to all the PhD students at LIP6 I have enjoyed collaborating, working, chatting, drinking with. In particular, thanks to my numerous office-mates: Marc-Ismaël Akodjenou, Jean-Noël Vittaud, Jean-Francois Pessiot, *Herr* Alex Spengler, Francis Maes, Tri Minh Do, Vinh Truong, Rudy Sicard, Trinh Ahn Phuc, Guillaume Wisniewski, David Buffoni, Bruno Pradel, Yann Soulard, etc. Another round is for the rest of the MALIRE team, Thierry Artières, Vincent Guigue, Ludovic Denoyer and also Ghislaine Mary, Jacqueline LeBacquer, Christophe Bouder and all the administrative staff who eased a lot my stay at LIP6.

During my different internships at NEC Labs in Princeton, I have been lucky to interact within a friendly environment in a remarkable team. Special thanks to Akshay Vashist, Bing Bai, Chris Burger, Eric Cosatto, Damien Delhomme, Hans-Peter Graf, Iain Melvin, Marina Spivak, Mat Miller, Seyda Ertekin and Karen Smith.

I would like to sincerely thank my great parents *sans qui je ne serais sans doute pas là aujourd'hui* and my cool sister without who I would be wearing the same sweater every day. I also have a deep thought for the rest of my family and its members recently gone. Many additional thanks to the dynamic and supportive Kiener family.

Well, I finally would like to cheerfully acknowledge my few non-machine learning friends that still remain. Here's to you: Amélie, Elie, Fabiàn, Loig, Mathilde, Jean-Marc, Laurent, Anthony, *le Klub Poètes*, the devoted *ABS* members, and also many old PC lads.

Thanks Mélusine for giving me enough time to work peacefully.

Résumé

Internet ainsi que tous les moyens numériques modernes disponibles pour communiquer, s'informer ou se divertir génèrent des données en quantités de plus en plus importantes. Dans des domaines aussi variés que la recherche d'information, la bio-informatique, la linguistique computationnelle ou la sécurité numérique, des méthodes automatiques capables d'organiser, classifier, ou transformer des téraoctets de données apportent une aide précieuse.

L'apprentissage artificiel traite de la conception d'algorithmes qui permettent d'entraîner de tels outils à l'aide d'exemples d'apprentissage. Utiliser certaines de ces méthodes pour automatiser le traitement de problèmes complexes, en particulier quand les quantités de données en jeu sont insurmontables pour des opérateurs humains, paraît inévitable. Malheureusement, la plupart des algorithmes d'apprentissage actuels, bien qu'efficaces sur de petites bases de données, présentent une complexité importante qui les rend inutilisables sur de trop grandes masses de données. Ainsi, il existe un besoin certain dans la communauté de l'apprentissage artificiel pour des méthodes capables d'être entraînées sur des ensembles d'apprentissage de grande échelle, et pouvant ainsi gérer les quantités colossales d'informations générées quotidiennement. Nous développons ces enjeux et défis dans le Chapitre 1.

Dans ce manuscrit, nous proposons des solutions pour réduire le temps d'entraînement et les besoins en mémoire d'algorithmes d'apprentissage sans pour autant dégrader leur précision. Nous nous intéressons en particulier aux Machines à Vecteurs Supports (SVMs), des méthodes populaires utilisées en général pour des tâches de classification automatique mais qui peuvent être adaptées à d'autres applications. Nous décrivons les SVMs en détail dans le Chapitre 2.

Ensuite, dans le Chapitre 3, nous étudions le processus d'apprentissage par descente de gradient stochastique pour les SVMs linéaires. Cela nous amène à définir et étudier le nouvel algorithme, SGD-QN. Après cela, nous introduisons une nouvelle procédure d'apprentissage: le principe du "Process/Reprocess". Nous déclinons alors trois algorithmes qui l'utilisent. Le Huller et LaSVM sont présentés dans le Chapitre 4. Ils servent à apprendre des SVMs destinés à traiter des problèmes de classification binaire (décision entre deux classes). Pour la tâche plus complexe de prédiction de sorties structurées, nous modifions par la suite en profondeur l'algorithme LaSVM, ce qui conduit à l'algorithme LaRank présenté dans le Chapitre 5. Notre dernière contribution concerne le problème récent de l'apprentissage avec une supervision ambigüe pour lequel nous proposons un nouveau cadre théorique (et un algorithme associé) dans le Chapitre 6. Nous l'appliquons alors au problème de l'étiquetage sémantique du langage naturel.

Tous les algorithmes introduits dans cette thèse atteignent les performances de l'état-de-l'art, en particulier en ce qui concerne les vitesses d'entraînement. La plupart d'entre eux ont été publiés dans des journaux ou actes de conférences internationaux. Des implantations efficaces de chaque méthode ont également été rendues disponibles. Dans la mesure du possible, nous décrivons nos nouveaux algorithmes de la manière la plus générale possible afin de faciliter leur application à des tâches nouvelles. Nous esquissons certaines d'entre elles dans le Chapitre 7.

Abstract

Internet as well as all the modern media of communication, information and entertainment entails a massive increase of digital data quantities. In various domains ranging from network security, information retrieval, to online advertisement, or computational linguistics automatic methods are needed to organize, classify or transform terabytes of numerical items.

Machine learning research concerns the design and development of algorithms that allow computers to learn based on data. A large number of accurate and efficient learning algorithms now exist and it seems rewarding to use them to automate more and more complex tasks, especially when humans have difficulties to handle large amounts of data. Unfortunately, most learning algorithms performs well on small databases but cannot be trained on large data quantities. Hence, there is a deep need for machine learning methods able to learn with millions of training instances so that they could enjoy the huge available data sources. We develop these issues in our introduction, in Chapter 1.

In this thesis, we propose solutions to reduce training time and memory requirements of learning algorithms while keeping strong performances in accuracy. In particular, among all the machine learning models, we focus on Support Vector Machines (SVMs) that are standard methods mostly used for automatic classification. We extensively describe them in Chapter 2

Throughout this dissertation, we propose different original algorithms for learning SVMs, depending on the final task they are destined to. First, in Chapter 3, we study the learning process of Stochastic Gradient Descent for the particular case of linear SVMs. This leads us to define and validate the new SGD-QN algorithm. Then we introduce a brand new learning principle: the Process/Reprocess strategy. We present three algorithms implementing it. The Huller and LaSVM are discussed in Chapter 4. They are designed towards training SVMs for binary classification. For the more complex task of structured output prediction, we refine intensively LaSVM: this results in the LaRank algorithm which is detailed in Chapter 5. Finally, in Chapter 6 is introduced the original framework of learning under ambiguous supervision which we apply to the task of semantic parsing of natural language.

Each algorithm introduced in this thesis achieves state-of-the-art performances, especially in terms of training speed. Almost all of them have been published in international peer-reviewed journals or conference proceedings. Corresponding implementations have also been released. As much as possible, we always keep the description of our innovative methods as generic as possible because we want to ease the design of any further derivation. Indeed, many directions can be followed to carry on with what we present in this dissertation. We list some of them in Chapter 7.

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

Contents

This thesis exhibits ways to exploit large-scale data sources in machine learning, especially for training Support Vector Machines. This introduction is designed to identify what were the motivations of this thesis and expose the main results we obtained. Section 1.1 sets up the background scenery and explains the pertinence of the new methods detailed in the next chapters. Afterward, Section 1.2 summarizes the different contributions that have been developed throughout this dissertation. The final section (Section 1.3) sketches the several chapters.

1.1 Large Scale Machine Learning

First of all, let us briefly present the general scientific domain of *machine learning* as well as some of its main applicative areas. We will then go on introducing the notion of large scale machine learning and explain its interests, the main issues it involves and therefore the reasons why working on it is relevant. This section ends by a discussion on the learning setup of online learning and a description of the specific scope of this thesis.

1.1.1 Machine Learning

The field of machine learning evolved from the broad field of *artificial intelligence*, which aims to mimic intelligent abilities of humans by machines. It is concerned with the design and development of algorithms that allow computers to learn based on data, such as from sensors or

databases. A major focus of machine learning research is to automatically learn to recognize complex patterns and take decisions based on data. Hence, machine learning is closely related to fields such as statistics, probability theory, data mining or pattern recognition.

Principle

In machine learning one considers the important question of how to make machines able to *learn*. Learning in this context is understood as inductive inference, where one observes examples that represent incomplete information about some statistical phenomenon. More specifically, an algorithm is said to learn with respect to a class of tasks, if its performance on this class of tasks increases with experience, given a measure of performance.

In this thesis, we only consider *supervised learning* problems. In such tasks, a machine learning algorithm induces a prediction function using a set of examples, called a *training set*. Each example consists of a pair formed by an observation annotated with a corresponding label. The goal of the learnt function is to predict the correct label associated with an observation. When the labels are discrete, the task is referred to as a classification problem. Otherwise, for real-valued labels, we speak of regression problems.

A learning algorithm must be able to perform correct predictions for observations belonging to the training set but also for unknown ones: machine learning is not only a question of remembering but also a matter of *generalizing* to unseen cases. In practice, a *testing set*, i.e. a set of examples never seen by the algorithm during training, along with a performance measure are thus employed to evaluate the generalization ability of a model.

Supervised learning is only a subfield of machine learning. For instance, one can consider unlabeled training examples and try to uncover hidden regularities or detect anomalies in the data: we then speak of unsupervised learning. One can also make use of both labeled and unlabeled data for training (typically a small amount of labeled data with a large amount of unlabeled data): this is referred to as semi-supervised learning.

Applications

Machine learning research is extremely active. A large number of accurate and efficient algorithms regularly arise. It seems then rewarding for scientists and engineers to learn how and where machine learning can be useful to automate tasks or provide predictions, especially when humans have difficulties to handle large amounts of data.

The long list of examples where machine learning techniques were successfully applied includes: Natural Language Processing (a vast field, see [Manning, 1999] for an overview), handwriting recognition (e.g. check reading [Le Cun *et al.*, 1997]), text categorization – spam filtering for example – (e.g. [Joachims, 2000]), bioinformatics (e.g. cancer tissue classification [Furey *et al.*, 2000]), network security (e.g. [Laskov *et al.*, 2004]), monitoring of electric appliances (e.g. [Murata and Onoda, 2002]), optimization of hard disk caching strategies [Gramacy *et al.*, 2003], drug discovery [Warmuth *et al.*, 2003], recommendation systems, natural scene analysis etc.

Of course, this brief summary is far from being complete. It focuses on supervised learning methods and does not mention applications of either unsupervised learning (e.g. clustering), or other branches of machine learning which extend its applicative range, but are not in the scope of this thesis.

1.1.2 Towards Large Scale Applications

The last decades have seen a massive increase of data quantities. In various domains such as biology, networking, or information retrieval, automatic methods, such as those that machine

Google	$>$ **1,000 billions**[1] indexed pages in July 2008
Flickr	$>$ **3 billions**[2] photos in late 2008
Wikipedia	$\approx$ **13 millions** articles in mid 2009
YouTube	$>$ **45 terabytes**[3] of videos in early 2007
Facebook	$>$ **200 millions**[4] active users in mid 2009
Twitter	$>$ **3.5 millions**[5] active users in mid 2009
E-mail spams	$\approx$ **100 billions**[6] per day in June 2007

[1] `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`
[2] `http://www.techcrunch.com/2008/11/03/three-billion-photos-at-flickr`
[3] `http://www.businessintelligencelowdown.com/2007/02/top_10_largest_.html`
[4] `http://www.facebook.com/press/info.php?statistics`
[5] `http://twitdir.com/`
[6] `http://www.spamunit.com/spam-statistics/`

Table 1.1: **Rough estimates of data resources of common Web services.** From the indexed pages of Google to the users of Facebook, many sources produce massive data quantities that need to be classified, organized, hierarchised, etc.

learning can provide, are needed to organize, classify or transform thousands of pieces of information. As illustration, Table 1.1 depicts the huge amounts of data generated and/or managed by some common Web services.

Computing Resources and Data Volume

Electronic computers have vastly enhanced our ability to compute complicated statistical models. As computing resources increases exponentially, one might think that no special care has to be taken to handle large-scale databases: the increase of processor speed would, eventually, make any algorithm tractable on any database, regardless of its size. A quick look at rough estimates proves this wrong.

As predicted by Moore's law, the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years since the 60's. This is depicted on Figure 1.1 for the period 1980-2010 (red curve) and reflects the exponential increase of computing power. But, on the other hand, since the 80's, hard-drive storage capacities empirically double every 18 months, more or less[1] as shows the blue curve of Figure 1.1.

It appears that data sizes outgrow computer speed. Cheap, pervasive and networked computers are now allowing to collect and store observations faster than to analyse them. Even worse, most machine learning algorithms demand computational resources that grow *much* faster than the volume of the data (the cost is usually at least quadratic).

Motivations of the Thesis

Any efficient learning algorithm should at least pay a brief look at each training example. There is a deep need for machine learning methods able to be trained on millions of training instances

[1] There is no law similar to Moore's law for hard-drive storage capacity. The informal Kryder's law states that disk area storage density doubles annually (`http://www.scientificamerican.com/article.cnfm?id=kryders-law`). But this appears to be mostly valid on the decade 1995-2005.

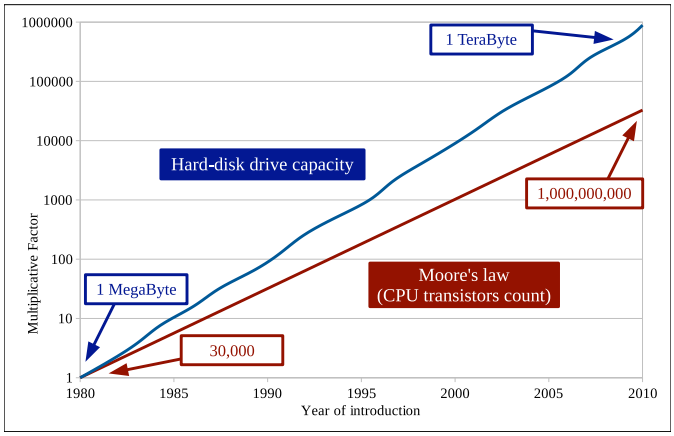


Figure 1.1: **Evolution of computing and storage resources.** Comparison of exponential growths of hard-disk drive capacity (blue) and CPU transistor counts (Moore's law) (red) against years of introduction. The logarithmic vertical axis represents their multiplicative factor since 1980. CPU counts double every 2 years while HD capacity empirically doubles every 18 months.

so that they could enjoy the massive recent databases. The main motivation of this thesis was then to improve the scalability of supervised learning techniques.

In short we have been seeking training algorithms with the following properties:

1. short training time (linear scaling w.r.t. training set size, if applicable),
2. low memory usage,
3. high generalization accuracy.

Of course, the work presented in this dissertation can not be applied to every machine learning field or application: it mostly relates to Support Vector Machines (SVMs). However, as we detail in Chapter 2, SVMs are a rather generic supervised machine learning framework that can be applied to lots of cases. That is the reason why we try to present most of our algorithms in a general way in order to ease the conception of derivations for new large-scale applications.

Supervised Large-scale Learning: an Heresy?

All the data sources displayed in Table 1.1 can not be directly used for supervised machine learning. Indeed, if one wants to learn a classifier for the 3 billions pictures of Flickr, these are not directly labeled with their topic. Same problem for the hundreds of billions of pages indexed by Google or for the loads of data generated by Facebook users. Manually annotating these to create data sets is a solution by far too complicated and costly. A pertinent question can thus be: is this useful to conceive methods for large-scale supervised learning if there is no large-scale annotated training set?

Fortunately, there exist tasks for which huge annotated training resources are available. A first example of productive source of labeled data is click-through information i.e. the sequence of clicks a user performs during an Internet session. Determining/classifying the future clicks of a

user is crucial for the online advertisement market and is a perfect machine learning application. Corresponding training data can be collected in huge quantities by Internet providers or Web services. In bioinformatics, for tasks such as DNA sequencing or protein classification, large amounts of supervised data can also be automatically gathered.

Furthermore, when the data is not directly labeled, the rising phenomenon of *collaborative labeling* can create new annotated corpora. In this case there is no direct annotation cost because all is performed by online users. For example, in the case of spam filtering, Email services receive millions of Email "marked as spam" everyday: these create perfect training examples for classification. Similarly, [Ma *et al.*, 2009] recently propose a work about the automatic detection of malicious URLs. Thanks to an Internet provider, they gathered more than 2 millions supervised training examples in a month. On picture sites like Flickr, users can tag their own pictures themselves: as a result, they create thousands of annotated examples for image retrieval (in July 2009, more than 6 millions photos were corresponding to the tag "beach" for example).

Collaborative labeling also provides huge annotated corpora for learning recommendation. Recommender systems are built to display information items (such as movies, music, books, etc.) that are likely of interest to a user and can be learnt with machine learning techniques. Training sets for such systems are composed by sets of items and their ratings given by different users. Such ratings can be legion and are usually gathered for free by Web merchants such as Netflix or Amazon on their websites. Netflix recently organized a challenge to determine the best movie recommender system:[2] they provided a training data set of around 100 millions ratings that over 480,000 users gave to nearly 18,000 movies.

This idea of collaborative annotation is even at the center of original *human-based computation* or *crowdsourcing* systems. For example, the *Game With A Purpose* project[3] targets to create online games which help creating supervised corpora (see [Von Ahn, 2006]) for tasks such as image recognition or segmentation, video retrieval, etc. Similarly, the *reCAPTCHA* system[4] produces annotated examples for Optical Character Recognition using special captchas[5] [Von Ahn *et al.*, 2008]. Annotating any kind of large data source with a reduced cost becomes credible.

All the above examples prove the existence of large-scale supervised data sources and exhibit the pertinence of the work described in this thesis. If still needed, the relevance of supervised large-scale machine learning is also assessed by the recent Pascal large-scale learning challenge [Sonnenburg *et al.*, 2008] which was entirely centered toward supervised learning.

1.1.3 Online Learning

In machine learning, the learning process defines how examples are used during the training phase. Most contributions of this dissertation are closely related to the *online learning* process because this is usually a suitable way of handling big training databases. This section then presents online learning and discusses its advantages and drawbacks.

Batch Learning

The standard way for learning the prediction function destined to any supervised machine learning task, is called *batch learning*. This training phase employs all the training examples together. First, a *cost function* measures and averages how well (or how poorly) the prediction system performs on all examples. According to this performance barometer, a global optimization step

[2] `http://www.netflixprize.com/`

[3] `http://www.gwap.com`

[4] `http://recaptcha.net/`

[5] A captcha is a type of challenge-response test used in computing to ensure that the response is not generated by a computer.

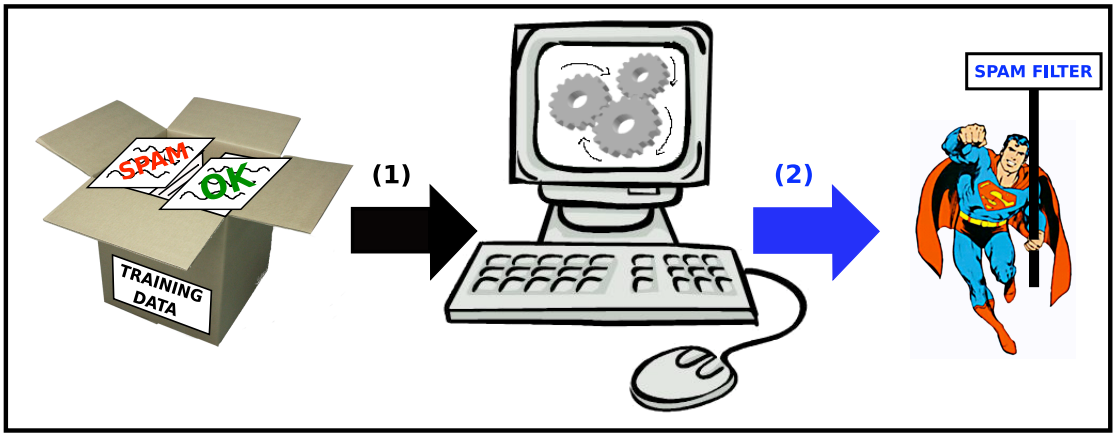


Figure 1.2: **Batch learning of spam filtering.** A training set of spam/non-spam documents is provided (left). **(1)** The learning algorithm (center) takes *the whole data set* as input. This requires a lot of memory and computational power. **(2)** After the (possibly long) training phase, a spam filter (right) learnt from the data is outputted. This is the unique solution if the problem is convex.

is performed on the parameters of the function. Such optimization steps are conducted until a pre-defined stopping condition is fulfilled. If the learning problem is convex (as it is for SVMs), the algorithm stops when the function parameters have converged to the unique solution of the problem. A rough illustration is given in Figure 1.2 for the case of learning an automatic spam filter. Examples of batch optimizers are Gradient Descent, Newton's method (see [Boyd and Vandenberghe, 2004] for details) or (L)BFGS [Nocedal, 1980]. They are popular because they are usually very accurate and can be fast, as long as the training set is not too big.

However, in many domains, data now arrives faster than batch methods are able to learn from it. Indeed, computing an average cost on all training instances takes a time (and memory) growing faster than the training set size and this is intractable on large scale data sets. To avoid wasting this data, one must switch from this traditional approach to systems that are able to mine continuous, high-volume, open-ended data streams as they arrive.

Online Learning

Online algorithms such as the Perceptron [Rosenblatt, 1958] have received a considerable interest for large-scale applications because they appear to perform well with comparatively small computational requirements (e.g. [Crammer and Singer, 2001, Collins and Roark, 2004]). The learning process of such algorithms is schematized in Figure 1.3. They perform a parameter update whenever they receive a fresh example (that can come from a closed set or a stream) and then discard it. Such methods are cheap in computations and memory as they only require to store and process a single example at a time.

Strong generalization guarantees for online algorithms can be obtained by assuming that each example is processed only once [Graepel *et al.*, 2000]. Indeed, before its corresponding parameter update, the performance of the learning system on each example reflects what has been learnt from the previous examples only and therefore can be interpreted as a measure of generalization (e.g. [Cesa-Bianchi and Lugosi, 2006]). Despite these theoretical guarantees, online algorithms

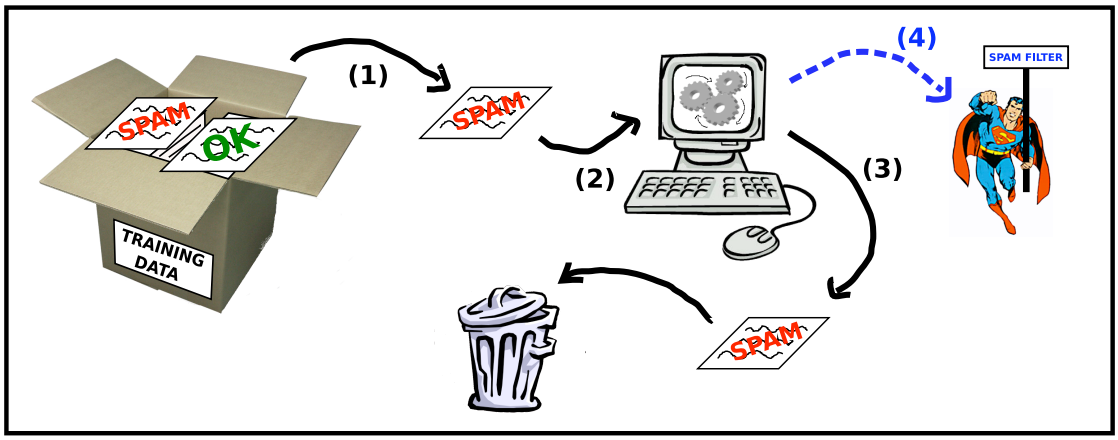


Figure 1.3: **Online learning of spam filtering.** A training set of spam/non-spam documents is provided (far left). **(1)** At each iteration, a training example is drawn from it. **(2)** The learning algorithm (center) takes *this single example* as input (low memory and computational power requirements). **(3)** After a learning step on it, this example is removed from the training set. The procedure (1)-(2)-(3) is carried-out until the training set is empty. **(4)** Anytime during the learning process, one has access to the current learnt spam filter, but it is not optimal.

rarely approach the generalization abilities of equivalent batch algorithms after a single pass. The solution is then to perform multiple passes on the training set. This achieves fair performances in practice (e.g. [Freund and Schapire, 1998]) but ruins the generalization guarantees and also increases a lot computational and memory requirements of online learning.

During this thesis, we have been seeking to produce learning algorithms sharing speed and scalability of online methods and generalization ability of batch techniques.

1.1.4 Scope of this Thesis

Among the wide range of tasks encompassed by supervised machine learning, this thesis is centered around two of them: classification and structured output prediction.

To address these problems, we have developped methods inspired by online learning to train Support Vector Machines in large-scale setups. Chapter 2 provides more insights on SVMs. In particular, Section 2.1 is entirely devoted to describe their application to classification and review the related standard algorithms. And Section 2.2 details how SVMs can be adapted to perform structured output prediction by following the approach proposed by [Tsochantaridis *et al.*, 2005], and how this formulation can be trained. But first, let us now introduce the two main tasks tackled in the remaining of this thesis.

Classification

In classification, one trains methods able to distinguish between different instances by assigning them a class label. In most cases there are two possible labels, we then speak of binary classification. Otherwise, it is called multiclass classification. Examples of instances are human faces, text documents, handwritten letters or digits, speech records, DNA sequences, etc.

An instance is described by its features, that are the characteristics of the examples for a given problem. For example, in handwriting recognition, an instance can be a black and white

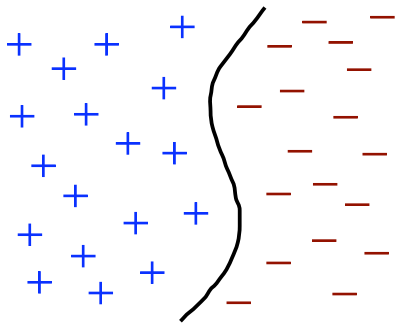


Figure 1.4: **Classification.** A binary classifier is a decision boundary (black line) which separates the mapping of training examples belonging to two sets (represented here by blue crosses and red minuses).

picture representing a symbol and its features the gray level of each of its pixels. Thus, the input to a classification task can equivalently be viewed as a two-dimensional matrix, whose axes are the examples and the features.

Classification can be divided into several sub-tasks:

1. data collection and representation,
2. feature selection and/or feature reduction,
3. data mapping and final decision.

Data collection and representation are mostly problem-specific. Therefore it is difficult to give general statements about this step of the process. Feature selection and feature reduction attempt to reduce the dimensionality (i.e. the number of features) for the classification step. This is not always essential or is implicitly performed in the third step.

Our work concentrates on learning the final classifier i.e. the process which finds a mapping between instances and labels. This final classifier is defined by the decision surface lying at the boundary between the mappings of the examples of each class. This is illustrated on Figure 1.4.

Structured Output Prediction

Much of the early research on supervised machine learning has focused on problems like classification and regression, where the prediction is a single univariate variable. However recent problems arise, requiring to predict complex objects like trees, sequences, or alignments. Many prediction problems can easily be broken into multiple binary classification problems, but other problems require an inherently structured prediction.

Consider, for example, the problem of semantic role labeling. For a given *input* sentence x, the goal is to predict the correct *output* parse tree y that reflects the semantic structure of the sentence. This is illustrated on the right-hand side of Figure 1.5. Training data of sentences that are labeled with the correct tree is available (e.g. from the Penn ProbBank [Kingsbury and Palmer, 2002]), making this prediction problem accessible for supervised learning. Compared to binary classification, the problem of predicting compound and structured outputs differs mainly by the choice of the outputs y, much more complex than simple atomic labels.

Here are some examples of structures commonly used as well as concrete applications (see [Bakır *et al.*, 2007] for a complete review of the field):

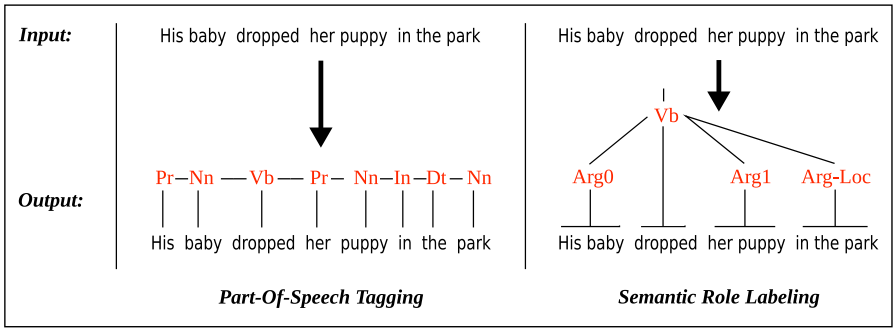


Figure 1.5: **Examples of structured output prediction tasks in Natural Language Processing.** *Left:* Part-of-speech tagging associates an input natural language sentence (top) with a sequence of part-of-speech tags such as Noun (Nn), verb (Vb), etc. (The output structure is a sequence.) *Right:* Semantic role labeling associates an input natural language sentence (top) to a tree connecting each verb with its semantic arguments. (The output structure is a tree.)

- *Sequences*: A standard *sequence labeling* problem is part-of-speech tagging. Given a sentence x represented as a sequence of words, the task is to predict the correct part-of-speech tag (e.g. noun or determiner) for each word (see the left-hand of Figure 1.5). Even if this problem could be formulated as a multiclass classification task for each word, predicting the sequence at a whole allows exploiting dependencies between tags (e.g. it is unlikely to see a verb after a determiner).
- *Trees*: We have already discussed the problem of semantic role labeling (Figure 1.5 (right)).
- *Alignments*: For comparative protein structure modelling, it is necessary to predict how the sequence of a new protein with unknown structure aligns against another sequence with known structure.

1.2 New Efficient Algorithms for Support Vector Machines

We now detail the contributions to the field of large-scale machine learning proposed in this dissertation. They can be split in three different pieces: (1) a novel generic algorithmic scheme for conceiving online SVMs solvers which have been successfully applied to classification and structured output prediction, (2) a quasi-Newton stochastic gradient algorithm for linear binary SVMs, (3) a method for learning SVMs under ambiguous supervision. Most of them have been the object of peer-reviewed publications in international journals or conference proceedings (see Appendix A).

1.2.1 A New Generation of Online SVM Dual Solvers

We present a new kind of solver for the dual formulation of SVMs. This contribution is actually threefold and takes up the main part of this thesis: it is the topic of both Chapter 4 and Chapter 5 (and also Appendix B).

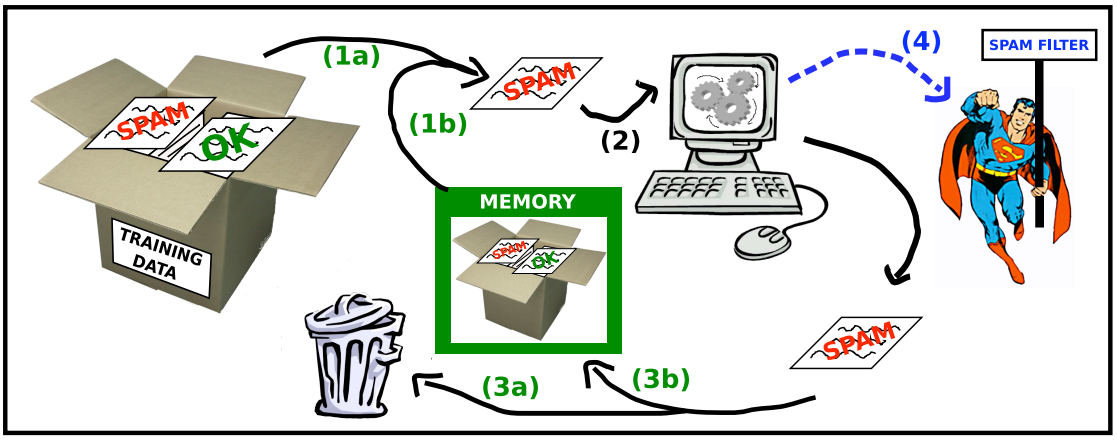


Figure 1.6: **Learning with the Process/Reprocess principle** Compared to a standard online process, an additional memory storage is added (green square). **(1)** At each iteration, a training example is either drawn from the training set ((1a) *process*) or from the additional memory ((1b) *reprocess*). **(2)** The learning algorithm (center) takes this single example as input. **(3)** After a learning step on it, this example is either discarded (3a) or stored in the memory (3b). The procedure (1)-(2)-(3) is carried-out until the training set is empty. **(4)** Anytime, one can have access to the current learnt spam filter.

The Process/Reprocess Principle

These new algorithms perform an online optimization of the dual objective of Support Vector Machines based on a so-called *process/reprocess principle*: when receiving a new example, they perform a first optimization step similar to that of a common online algorithm. In addition to this Process operation, they perform Reprocess operations: each of which is a basic optimization step applied to randomly chosen previously seen training examples. Figure 1.6 illustrates this learning scheme. The Reprocess operations force these algorithms to store a fraction of the training examples to re-visit them now and then. This causes extra-storing and extra-computations compared to standard online algorithms: these methods are not strictly online.[6] However these training algorithms still scale better than batch methods because the number of stored examples is usually much smaller then the training set size.

This alternative online behavior presents interesting properties, especially for large-scale applications. Indeed, results provided in this dissertation show that online optimization with the Process/Reprocess principle leads to algorithms providing fair approximate solutions on the whole course of learning and achieving good accuracies while having low computational costs.

Family of Algorithms

During this thesis, we successively applied the Process/Reprocess principle to several concrete problems. Hence, we developed a whole family of efficient algorithms.

Chapter 4 introduces two Process/Reprocess algorithms for binary classification. Named the Huller and LaSVM, they yield competitive misclassification rates after a single pass over the training examples, outspeeding state-of-the-art SVMs solvers. LaSVM outperforms the Huller

[6] Yet we sometimes refer to these as online algorithms in this thesis: it is a common naming abuse.

because it handles noisy data in a better way. We also show how *active example selection* can yield even faster training, higher accuracies, and simpler models, using only a fraction of the training examples. Chapter 5 then proposes an online solver of the dual formulation of SVMs for structured output prediction. The LaRank algorithm, implementing the Process/Reprocess principle, is applied to the tasks of multiclass classification and sequence labeling. In both cases, LaRank shares the generalization performances of batch optimizers and the speed of standard online methods.

Theoretical Study

Every derivation is proved to eventually converge to the same solution as batch methods by theoretical proofs spread in the chapters.

Moreover, in Section 4.4, we provide a theoretical study of the Process/Reprocess principle in the context of *online approximate optimization*. We analyse a simple algorithm for SVMs for binary classification, and show that a constant number of Reprocess operations is sufficient to maintain, on the course of the algorithm, an averaged accuracy criterion, with a computational cost that scales as well as the best existing SVMs algorithms with the number of examples.

1.2.2 A Carefully Designed Second-Order SGD

Stochastic Gradient Descent is known to be a fast learning algorithm in the large-scale setup. In particular, numerous recent works report great performances for training linear SVMs.

In Chapter 3, we discuss how to train efficiently linear SVMs and propose SGD-QN: a stochastic gradient descent algorithm that makes careful use of second-order information and splits the parameter update into independently scheduled components. Thanks to this design, SGD-QN iterates nearly as fast as a first-order stochastic gradient descent but requires less iterations to achieve the same accuracy. This algorithm won the "Wild Track" of the first PASCAL Large Scale Learning Challenge [Sonnenburg *et al.*, 2008].

1.2.3 A Learning Method for Ambiguously Supervised SVMs

This contribution addresses the fresh problem of learning from *ambiguous supervision*, focusing on the task of semantic parsing. A learning problem is said to be ambiguously supervised when, for a given training input, a *set of output candidates* (rather than the only correct output) is provided with no prior of which one is correct. In Chapter 6 is then introduced a new reduction from ambiguous multiclass classification to the problem of noisy label ranking, which we then cast into a SVMs formulation. We propose an online algorithm for learning these SVMs. An empirical validation on semantic parsing data sets demonstrates the efficiency of this approach.

This contribution does not directly focus on large-scale learning. In particular, the related experiments concern small-size data sets. Yet, our contribution involves an online algorithm presenting good scaling properties towards large-scale problems.

Moreover, we believe this chapter is important because learning from ambiguous supervision will be a key challenge in the future. Indeed, the cost for producing ambiguously annotated corpora is far less than the one required for producing perfectly annotated ones. Large-scale ambiguously annotated data sets will be likely to appear in the next few years. Being able to properly use them would be rewarding.

1.2.4 Careful Implementations

For almost all the new algorithms discussed in this thesis, a corresponding efficient implementation (in C or C++) is freely available.[7] Even if this does not appear directly in the present dissertation, we consider this as a contribution. Indeed a careful implementation is a key factor when dealing with large amounts of data.

This issue is extensively discussed for the particular case of Stochastic Gradient Descent algorithms in Chapter 3. Some implementation details are also provided for all other algorithms.

1.3 Outline of the Thesis

The chapters are not arranged in chronological order but rather follow the increase in complexity of the different prediction models to be learnt. For interested readers, the chronological order in which the different pieces of work have been developed, is: Chapter 4, then Chapter 5, Chapter 3 and Chapter 6.

- **Chapter 2** presents the formalism of Support Vector Machines for classification and for structured output prediction. It also describes the main notations and details some of the state-of-the-art batch and online learning methods for SVMs.
- In **Chapter 3**, we study the learning process of Stochastic Gradient Descent for the particular case of linear SVMs. This leads us to define and validate the new SGD-QN algorithm.
- **Chapter 4** explains the Process/Reprocess principle via the simple Huller algorithm. We then analyse the LaSVM algorithm for solving binary classification, discuss the benefit of joining active and online learning, and present a lemma which assesses generalization abilities of the Huller and LaSVM.
- In **Chapter 5**, we discuss how to learn SVMs for structured output prediction with LaRank, an algorithm implementing the Process/Reprocess principle. Derivation to multiclass classification and sequence labeling are detailed.
- In **Chapter 6** is introduced the original framework of learning under ambiguous supervision which we apply to the structured task of semantic parsing.
- **Chapter 7** presents our concluding remarks and explores some future research directions.

Three supplements are proposed at the end of this dissertation:

- **Appendix A** catalogs the different publications regarding this thesis contributions.
- **Appendix B** addresses the convergence properties of algorithms discussed in Chapter 4.
- **Appendix C** is not directly related to this thesis. It presents some of our recent work on Natural Language Processing in which we experience some ways of *learning to disambiguate language* using world knowledge and neural networks.

[7] Codes have been released under the GPL3 license and can be downloaded either at `http://webia.lip6.fr/~bordes/mywiki/doku.php?id=codes` or from the `mloss.org` repository for machine learning open source softwares.

2

Support Vector Machines

Contents

In this thesis, we address the training of Support Vector Machines (SVMs) on large scale databases. SVMs [Vapnik, 1998] are supervised learning methods originally used for binary classification and regression. They are the successful application of the kernel idea [Aizerman *et al.*, 1964] to large margin classifiers [Vapnik and Lerner, 1963] and have proved to be powerful tools. Nowadays SVMs are used in various research and engineering areas ranging from breast cancer diagnosis, recommendation system, database marketing, or detection of protein homologies, to text categorization, or face recognition, etc.[1] The contributions of this dissertation cover the general framework of SVMs. Hence, their applicative scope is potentially very vast.

The present chapter introduces Support Vector Machines along with some state-of-the-art algorithms to train them. We do not claim to be exhaustive here, and we focus on the main methods of the literature that are the most related to our work. For more details, [Cristianini and Shawe-Taylor, 2000] propose a deep and comprehensive introduction to Support Vector Machines. Section 2.1 focuses on binary classification, the original application of SVMs. In particular, Section 2.1.2 presents batch SVMs training methods and Section 2.1.3 online kernel algorithms. Then, Section 2.2 introduces the recent application of SVMs to the case of structured output prediction following the work presented by [Tsochantaridis *et al.*, 2005]. Existing batch and online methods are finally discussed.

[1] The webpage `http://www.clopinet.com/isabelle/Projects/SVM/applist.html` displays many successful applications of SVMs.

2.1 Kernel Classifiers

Early kernel classifiers [Aizerman *et al.*, 1964] were derived from the perceptron [Rosenblatt, 1958], a simple and efficient online learning algorithm. They associate classes $y = \pm 1$ to patterns $x \in \mathcal{X}$ by first transforming the patterns into feature vectors $\Phi(x)$ and taking the sign of a linear discriminant function:

$$f(x) = \langle w, \Phi(x) \rangle + b \tag{2.1}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product in the feature space endowed by $\Phi(\cdot)$. The parameters w and b are determined by running some learning algorithm on a set of training examples $(x_1, y_1) \cdots (x_n, y_n)$. These classifiers are called Φ-machines, their feature function Φ is usually hand chosen for each particular problem [Nilsson, 1965]. [Aizerman *et al.*, 1964] transform such linear classifiers by leveraging two theorems of the *Reproducing Kernel* theory [Aronszajn, 1950].

The *Representation Theorem* states that many Φ-machine learning algorithms produce parameter vectors w that can be expressed as a linear combinations of the training patterns.

$$w = \sum_{i=1}^{n} \alpha_i \Phi(x_i)$$

The linear discriminant function (2.1) can then be written as a *kernel expansion*:

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) + b \tag{2.2}$$

where the *kernel* function $k(x, \bar{x})$ represents the dot products $\langle \Phi(x), \Phi(\bar{x}) \rangle$ in feature space. This expression is most useful when a large fraction of the coefficients α_i are zero. Examples such that $\alpha_i \neq 0$ are then called *Support Vectors*.

Mercer's Theorem precisely states which kernel functions correspond to a dot product for some feature space. Kernel classifiers deal with the kernel function $k(x, \bar{x})$ without explicitly using the corresponding feature function $\Phi(x)$. Common kernel involve the simplest *linear kernel* $k(x, \bar{x}) = \langle x, \bar{x} \rangle$, the *polynomial kernel* $k(x, \bar{x}) = (1 + \langle x, \bar{x} \rangle)^p$ (where the positive integer p is the degree) and the well-known *RBF kernel* $k(x, \bar{x}) = e^{-\gamma \|x - \bar{x}\|^2}$ (with $\gamma > 0$) which defines an implicit feature space of infinite dimension.

Kernel classifiers handle such large feature spaces with the comparatively modest computational costs of the kernel function. On the other hand, kernel classifiers must control the decision function complexity in order to avoid overfitting the training data in such large feature spaces. This can be achieved by keeping the number of support vectors as low as possible [Littlestone and Warmuth, 1986] or by searching decision boundaries that separate the examples with the largest margin [Vapnik and Lerner, 1963, Vapnik, 1998].

2.1.1 Support Vector Machines

Support Vector Machines were defined by three incremental steps. First, [Vapnik and Lerner, 1963] propose to construct the *Optimal Hyperplane*, that is, the linear classifier that separates the training examples with the widest margin. Then, [Guyon *et al.*, 1993] propose to construct the Optimal Hyperplane in the feature space induced by a kernel function. Finally, [Cortes and Vapnik, 1995] show that noisy problems are best addressed by allowing some examples to violate the margin constraint.

The idea of the maximization comes from the following reasoning. As for early classifiers, predictions are carried out by taking the sign of the function f defined in (2.2). Geometrically,

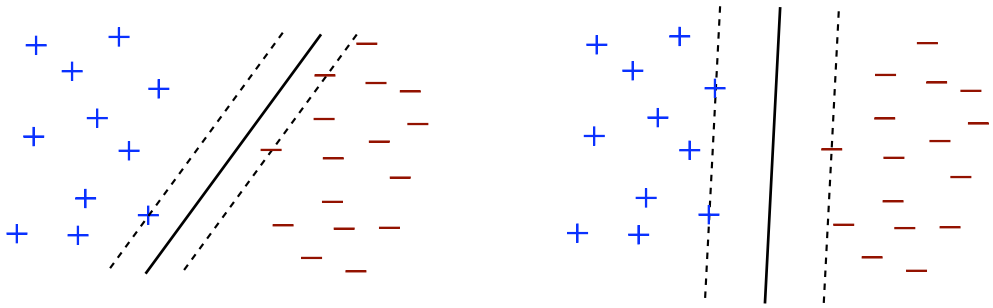


Figure 2.1: **Margins.** Two hyperplanes for separating crosses (blue) and minuses (red). *Left*: hyperplane with a small margin. *Right*: hyperplane with a large margin. The margin is the distance between the two dashed hyperplanes. SVMs are classifiers maximizing the margin.

the equation $f(x) = 0$ actually defines an hyperplane in the space induced by the feature function $\Phi(x)$. It is depicted as a black line in Figure 2.1. In the SVM framework, this hyperplane is enforced to separate the two classes of examples with the largest margin because, intuitively, a classifier with a larger margin is more noise-resistant. This can be expressed by the following set of constraints:

$$\forall i\,, \quad \begin{cases} f(x_i) \geq \gamma & \text{if } y_i = +1 \\ f(x_i) \leq -\gamma & \text{if } y_i = -1 \end{cases} \tag{2.3}$$

with γ an arbitrary positive tolerance. By rescaling w and b, we can set $\gamma = 1$, with no loss of generality, and group the above constraints in a single formula

$$\forall i\,, \quad y_i\, f(x_i) \geq 1\ . \tag{2.4}$$

The margin is defined as the distance between the hyperplanes $f(x) = 1$ and $f(x) = -1$ (dashed lines in Figure 2.1). A straightforward calculus provides its analytical value

$$\text{margin} = \frac{2}{||w||}\ . \tag{2.5}$$

Finally, Support Vector Machines minimize the following objective function in feature space.

$$\min_{w,b}\ P(w,b) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \ell(y_i\, f(x_i)) \tag{2.6}$$

The first term of the equation expresses the maximization of the margin (2.5). The second term enforces to satisfy the constraints (2.4). Indeed the function ℓ, named the hinge loss, is defined as $\ell(y_i\, f(x_i)) = \max(0, 1 - y_i\, f(x_i))$ and is directly related to the constraints set. The hinge loss can also be seen as an intuitive measure of the quality of the classifier f on each training example (x_i, y_i): the larger $\ell(y_i\, f(x_i))$ is, the worse the classifier performs on (x_i, y_i).

Introducing the slack variables ξ_i, one usually gets rid of the inconvenient max of the loss and rewrite the problem as

$$\min_{w,b}\ P(w,b) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{with} \quad \begin{cases} \forall\, i & y_i\, f(x_i) \geq 1 - \xi_i \\ \forall\, i & \xi_i \geq 0 \end{cases} \tag{2.7}$$

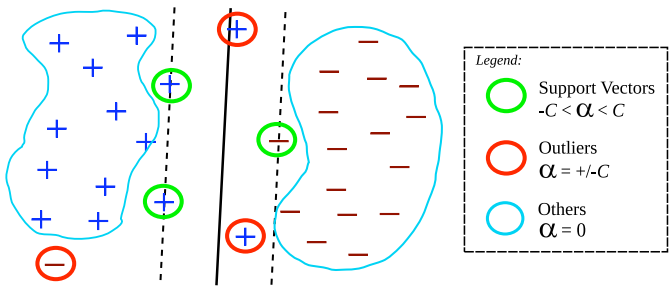


Figure 2.2: **Separating hyperplane and dual coefficients.** Support vectors are the examples on which lies the margin and correspond to non-zero α. The C parameter is essential to bound the α of misclassified instances (outliers) and yet lower their influence in the solution.

For very large values of the hyper-parameter C, this expression minimizes $\|w\|$ (i.e. maximizes (2.5)) under the constraint that all training examples are correctly classified with a loss $\ell(y_i\ f(x_i))$ equal to zero. This is termed the Hard Margin case. Smaller values of C relax this constraint and give the so-called Soft Margin SVMs that produces markedly better results on noisy problems [Cortes and Vapnik, 1995]. SVMs have been very successful and are very widely used because they reliably deliver state-of-the-art classifiers with minimal tweaking.

In practice learning SVMs can be achieved by solving the dual of this convex optimization problem. The coefficients α_i of the SVM kernel expansion (2.2) are found by defining the dual objective function

$$D(\boldsymbol{\alpha}) = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j) \tag{2.8}$$

and solving the SVM dual *Quadratic Programming* (QP) problem.

$$\max_{\boldsymbol{\alpha}} \ D(\boldsymbol{\alpha}) \quad \text{with} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = 0 \\ A_i \le \alpha_i \le B_i \\ \quad A_i = \min(0, C y_i) \\ \quad B_i = \max(0, C y_i) \end{array} \right. \tag{2.9}$$

Figure 2.2 illustrates how the separating hyperplane and the margin are related to the final coefficients α_i. As stated by the representation theorem, the discriminant function can be expressed as a kernel expansion (2.2) involving only a fraction of the training examples, those corresponding to non-zero α, i.e. the support vectors.

The formulation (2.9) slightly deviates from the standard formulation [Cortes and Vapnik, 1995] because it makes the α_i coefficients positive when $y_i = +1$ and negative when $y_i = -1$. The standard formulation enforcing all α_i to be positive is defined as:

$$D(\boldsymbol{\alpha}) = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \quad \text{with} \quad \left\{ \begin{array}{l} \sum_i \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \end{array} \right. \tag{2.10}$$

Both formulations lead to the same solution. In most of this thesis, we work with the dual QP (2.9). (Only in Section 4.4, we use (2.10) because it provides more convenient notations.)

Computational Cost of SVMs There are two intuitive lower bounds on the computational cost of any algorithm able to solve the SVM QP problem for arbitrary matrices $k_{ij} = k(x_i, x_j)$.

1. Suppose that an oracle reveals whether $\alpha_i = 0$ or $\alpha_i = \pm C$ for all $i = 1 \dots n$. Computing the remaining $0 < |\alpha_i| < C$ amounts to inverting a matrix of size $R \times R$ where R is the number of support vectors such that $0 < |\alpha_i| < C$. This typically requires a number of operations proportional to R^3.

2. Simply verifying that a vector $\boldsymbol{\alpha}$ is a solution of the SVM QP problem involves computing the gradients of $D(\boldsymbol{\alpha})$ and checking the Karush-Kuhn-Tucker optimality conditions [Vapnik, 1998]. With n examples and S support vectors, this requires a number of operations proportional to $n\,S$.

Few support vectors reach the upper bound C when it gets large. The cost is then dominated by the $R^3 \approx S^3$. Otherwise the term $n\,S$ is usually larger. The final number of support vectors therefore is the critical component of the computational cost of the SVM QP problem.

Assume that increasingly large sets of training examples are drawn from an unknown distribution $P(x, y)$. Let $\mathcal{B}$ be the error rate achieved by the best decision function (2.1) for that distribution. When $\mathcal{B} > 0$, [Steinwart, 2004] shows that the number of support vectors is asymptotically equivalent to $2n\mathcal{B}$. Therefore, regardless of the exact algorithm used, the asymptotic computational cost of solving the SVM QP problem grows at least like n^2 when C is small and n^3 when C gets large. Empirical evidence shows that modern SVM solvers [Chang and Lin, 2001 2004, Collobert and Bengio, 2001] come close to these scaling laws.

Practice however is dominated by the constant factors. When the number of examples grows, the kernel matrix $k_{ij} = k(x_i, x_j)$ becomes very large and cannot be stored in memory. Kernel values must be computed on the fly or retrieved from a cache of often accessed values. When the cost of computing each kernel value is relatively high, the kernel cache hit rate becomes a major component of the cost of solving the SVM QP problem [Joachims, 1999]. Large problems must be addressed by using algorithms that access kernel values with very consistent patterns.

2.1.2 Solving SVMs with SMO

Efficient batch numerical algorithms have been developed to solve the SVM QP problem (2.9). The best known methods are the Conjugate Gradient method [Vapnik, 1982, pages 359–362] and the Sequential Minimal Optimization (SMO) [Platt, 1999]. Both methods work by making successive searches along well chosen directions. Some famous SVM solvers like SVMLight [Joachims, 1999] or SVMTorch [Collobert and Bengio, 2001] propose to use decomposition algorithms to define such directions. This section mainly details SMO, as this is our main reference SVM solver in this thesis. In particular, we compare our methods with the state-of-the-art implementation of SMO, LibSVM [Chang and Lin, 2001 2004]. For a complete review of efficient batch SVM solvers see [Bottou and Lin, 2007].

Sequential Direction Search

Each direction search solves the restriction of the SVM problem to the half-line starting from the current vector $\boldsymbol{\alpha}$ and extending along the specified direction $\boldsymbol{u}$. Such a search yields a new feasible vector $\boldsymbol{\alpha} + \lambda^* \boldsymbol{u}$.

$$\lambda^* = \arg\max D(\boldsymbol{\alpha} + \lambda \boldsymbol{u}) \quad \text{with} \quad 0 \le \lambda \le \phi(\boldsymbol{\alpha}, \boldsymbol{u}) \tag{2.11}$$

The upper bound $\phi(\boldsymbol{\alpha}, \boldsymbol{u})$ ensures that $\boldsymbol{\alpha} + \lambda \boldsymbol{u}$ is feasible as well.

$$\phi(\boldsymbol{\alpha}, \boldsymbol{u}) \;=\; \min \left\{ \begin{array}{cl} 0 & \text{if } \sum_k u_k \neq 0 \\ (B_i - \alpha_i)/u_i & \text{for all } i \text{ such that } u_i > 0 \\ (A_j - \alpha_j)/u_j & \text{for all } j \text{ such that } u_j < 0 \end{array} \right\} \tag{2.12}$$

Calculus shows that the optimal value is achieved for

$$\lambda^* = \min \left\{ \phi(\boldsymbol{\alpha}, \boldsymbol{u}) \;,\; \frac{\sum_i g_i \, u_i}{\sum_{i,j} u_i u_j \, k_{ij}} \right\} \tag{2.13}$$

where $k_{ij} = k(x_i, x_j)$ and $\boldsymbol{g} = (g_1 \ldots g_n)$ is the gradient of $D(\boldsymbol{\alpha})$:

$$g_k \;=\; \frac{\partial D(\boldsymbol{\alpha})}{\partial \alpha_k} \;=\; y_k - \sum_i \alpha_i k(x_i, x_k) \;=\; y_k - f(x_k) + b \;. \tag{2.14}$$

Sequential Minimal Optimization

[Platt, 1999] observes that direction search computations are much faster when the search direction $\boldsymbol{u}$ mostly contains zero coefficients. At least two coefficients are needed to ensure that $\sum_k u_k = 0$. The *Sequential Minimal Optimization* (SMO) algorithm uses search directions whose coefficients are all zero except for a single $+1$ and a single -1.

Practical implementations of the SMO algorithm [Chang and Lin, 2001 2004, Collobert and Bengio, 2001] usually rely on a small positive tolerance $\tau > 0$. They only select directions $\boldsymbol{u}$ such that $\phi(\boldsymbol{\alpha}, \boldsymbol{u}) > 0$ and $\langle \boldsymbol{u}, \boldsymbol{g} \rangle > \tau$. This means that we can move along direction $\boldsymbol{u}$ without immediately reaching a constraint and increase the value of $D(\boldsymbol{\alpha})$. Such directions are defined by the so-called *τ-violating pair* (i, j):

$$(i,j) \text{ is a } \tau\text{-violating pair} \iff \left\{ \begin{array}{c} \alpha_i < B_i \\ \alpha_j > A_j \\ g_i - g_j > \tau \end{array} \right.$$

Algorithm 1 SMO Algorithm

1: Set $\boldsymbol{\alpha} \leftarrow \mathbf{0}$ and compute the initial gradient $\boldsymbol{g}$ (equation 4.2)
2: Choose a τ-violating pair (i, j). Stop if no such pair exists.
3: $\lambda \leftarrow \min \left\{ \dfrac{g_i - g_j}{k_{ii} + k_{jj} - 2k_{ij}},\; B_i - \alpha_i,\; \alpha_j - A_j \right\}$
$\alpha_i \leftarrow \alpha_i + \lambda \;,\;\; \alpha_j \leftarrow \alpha_j - \lambda$
$g_s \leftarrow g_s - \lambda(k_{is} - k_{js}) \quad \forall\, s \in \{1 \ldots n\}$
4: Return to step 2.

Algorithm 1 sketches SMO but does not specify how exactly the τ-violating pairs are chosen. Modern implementations of SMO select the τ-violating pair (i, j) that maximizes the directional gradient $\langle \boldsymbol{u}, \boldsymbol{g} \rangle$. This choice was described in the context of Optimal Hyperplanes in both [Vapnik, 1982, pages 362–364] and [Vapnik *et al.*, 1984].

Regardless of how exactly the τ-violating pairs are chosen, [Keerthi and Gilbert, 2002] assert that the SMO algorithm stops after a finite number of steps. This assertion is correct despite a slight flaw in their final argument [Takahashi and Nishi, 2003]. When SMO stops, no τ-violating pair remain. The corresponding $\boldsymbol{\alpha}$ is called a *τ-approximate solution*. Proposition 23 in Appendix B establishes that such approximate solutions indicate the location of the solution(s) of the SVM QP problem when the tolerance τ become close to zero.

2.1.3 Online Kernel Classifiers

On large-scale problems, batch methods solving the SVM QP problem exactly become intractable. Even when they implement efficient caching procedures to avoid multiple costly calculations of kernel values, their computational requirements overcome computing resources.

Hence, many authors have sought to replicate the SVM success with an online learning process by applying the large margin idea to some simple online algorithms [Freund and Schapire, 1998, Frieß *et al.*, 1998, Gentile, 2001, Li and Long, 2002, Crammer and Singer, 2003]. These methods present better scaling properties than batch ones but they do not actually solve the SVM QP. As a consequence, they usually suffer a loss of generalization. However, on many large-scale applications they are the only methods available.

Kernel Perceptrons

The earliest online kernel classifiers [Aizerman *et al.*, 1964] were derived from the Perceptron algorithm [Rosenblatt, 1958]. The decision function (2.2) is represented by maintaining the set $\mathcal{S}$ of the indices i of the support vectors. The bias parameter b remains zero. We depict the kernel perceptron in Algorithm 2.

Algorithm 2 Kernel Perceptron

1: $\mathcal{S} \leftarrow \emptyset, \quad b \leftarrow 0.$
2: Pick a random example (x_t, y_t)
3: Compute $f(x_t) = \sum_{i \in \mathcal{S}} \alpha_i \, k(x_t, x_i) + b$
4: **if** $y_t \, f(x_t) \leq 0$ **then**
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{t\}, \quad \alpha_t \leftarrow y_t$
6: **end if**
7: Return to step 2.

Such *online learning* algorithms require far less memory than batch methods because the examples are processed one by one and can be discarded after being examined.

Iterations such that $y_t \, f(x_t) < 0$ are called *mistakes* because they correspond to patterns misclassified by the perceptron decision boundary. The algorithm then modifies the decision boundary by inserting the misclassified pattern into the kernel expansion. When a solution exists, Novikoff's theorem [Novikoff, 1962] states that the algorithm converges after a finite number of mistakes, or equivalently after inserting a finite number of support vectors. Noisy data sets are more problematic.

Large Margin Kernel Perceptrons

The success of Support Vector Machines has shown that large classification margins were desirable. On the other hand, the Kernel Perceptron (Section 2.1.3) makes no attempt to achieve large margins because it happily ignores training examples that are very close to being misclassified.

Many authors have proposed to close the gap with online kernel classifiers by providing larger margins. The Averaged Perceptron [Freund and Schapire, 1998] decision rule is the majority vote of all the decision rules obtained after each iteration of the Kernel Perceptron algorithm. This choice provides a bound comparable to those offered in support of SVMs. Other algorithms [Frieß *et al.*, 1998, Gentile, 2001, Li and Long, 2002, Crammer and Singer, 2003] explicitly construct larger margins. In particular, the *passive-aggressive* algorithm [Crammer *et al.*, 2006] (see Algorithm 3) performs updates when the margin $y_t \, f(x_t)$ of the freshly drawn example

is lower than 1, with a magnitude based on analytical solutions of simple constraint problems similar to QP (2.9).

Algorithm 3 Passive-Aggressive (C)

1: $\mathcal{S} \leftarrow \emptyset, \quad b \leftarrow 0.$
2: Pick a random example (x_t, y_t)
3: Compute $f(x_t) = \sum_{i \in \mathcal{S}} \alpha_i \, k(x_t, x_i) + b$
4: **if** $y_t \, f(x_t) \leq 1$ **then**
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{t\}, \quad \alpha_t \leftarrow y_t \min\left(C, \frac{1 - y_t \, f(x_t)}{k(x_t, x_t)}\right)$
6: **end if**
7: Return to step 2.

Hence, large margin algorithms modify the decision boundary whenever a training example is either misclassified or classified with an insufficient margin. Such examples are then inserted into the kernel expansion with a suitable coefficient. Unfortunately, this change significantly increases the number of mistakes and therefore the number of support vectors. The increased computational cost and the potential overfitting undermines the positive effects of the margin.

Kernel Perceptrons with Removal Step

This is why [Crammer *et al.*, 2004] suggest an additional step for *removing* support vectors from the kernel expansion (2.2). The Budget Perceptron (Algorithm 4) performs very nicely on relatively clean data sets.

Algorithm 4 Budget Kernel Perceptron (β, N)

1: $\mathcal{S} \leftarrow \emptyset, \quad b \leftarrow 0.$
2: Pick a random example (x_t, y_t)
3: Compute $f(x_t) = \sum_{i \in \mathcal{S}} \alpha_i \, k(x_t, x_i) + b$
4: **if** $y_t \, f(x_t) \leq \beta$ **then**
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{t\}, \quad \alpha_t \leftarrow y_t$
6: $\quad$ **if** $|\mathcal{S}| > N$ **then**
7: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} - \{ \arg\max_{i \in \mathcal{S}} \; y_i \, (f(x_i) - \alpha_i \, k(x_i, x_i)) \}$
8: $\quad$ **end if**
9: **end if**
10: Return to step 2.

Online kernel classifiers usually experience considerable problems with noisy data sets. Each iteration is likely to cause a mistake because the best achievable misclassification rate for such problems is high. The number of support vectors increases very rapidly and potentially causes overfitting and poor convergence. More sophisticated support vector removal criteria avoid this drawback [Weston *et al.*, 2005]. This modified algorithm outperforms all other online kernel classifiers on noisy data sets and approaches the performance of Support Vector Machines with less support vectors.

Incremental Algorithms

Unfortunately, even the most sophisticated kernel perceptrons achieve generalization accuracies lower than those of batch SVMs, because their online process make too scarce use of training

examples. Incremental algorithms [Cauwenberghs and Poggio, 2001, Laskov *et al.*, 2006] attempt to combine the precision of batch SVMs with a somewhat online training process.

At each time index, they perform a batch optimization of a SVM objective function restricted on the examples seen so far until reaching an optimality criterion. At each step, only one point is added to the training set and one recomputes the exact SVM solution of the whole data set seen so far. Hence, one does not consider a finite training set of size n anymore but a *succession* of training sets whose sizes increases by one at each step.

In this thesis, we denote $P_t(w)$ the primal cost function restricted to the set containing the first t examples.[2] An incremental algorithm thus solves recursively the following problems:

$$\min_{w,b} \; P_t(w,b) = \|w\|^2 + C\sum_{i=1}^{t}\xi_i \quad \text{with} \quad \left\{ \begin{array}{ll} \forall\, i = 1,\dots t & y_i\, f(x_i) \geq 1 - \xi_i \\ \forall\, i = 1,\dots t & \xi_i \geq 0 \end{array} \right. \tag{2.15}$$

Similarly $D_t(\boldsymbol{\alpha})$ denotes the associated dual objective. The SVM QP (2.9) becomes:

$$\max_{\boldsymbol{\alpha}} \; D_t(\boldsymbol{\alpha}) = \sum_{i=1}^{t}\alpha_i - \frac{1}{2}\sum_{i,j\leq t} y_i y_j \alpha_i \alpha_j k(x_i,x_j) \quad \text{with} \quad \left\{ \begin{array}{ll} \sum_i \alpha_i = 0 & \\ \forall\, i = 1,\dots t & 0 \leq \alpha_i \leq C \end{array} \right. \tag{2.16}$$

Incremental algorithms are mostly used either in active learning, or, in an incremental/decremental setting, to compute leave-one-out errors. Such methods requires very efficient implementation to be competitive, in particular, ways to avoid to re-computing the whole solution from scratch at each step are crucial.

The condition to remain optimal at every step means that an incremental algorithm has to test and potentially train on every instances seen so far: this is intractable on large training sets. SimpleSVM [Vishwanathan *et al.*, 2003] is derived from the incremental setup but uses a loose optimality criterion only requiring to be optimal on a subset of examples, and thus scales better.

2.1.4 Solving Linear SVMs

The use of a linear kernel heavily simplifies the SVM optimization problem. Indeed such a kernel allows to explicitly express the parameter vector w. This means (i) no need to use a kernel expansion as in (2.2) anymore, and (ii) no need to store or compute the kernel matrix. Computing gradients of either the primal or dual cost function is cheap and depends only on the sparsity of the instances. The use of linear kernel is thus very interesting when one needs to handle large-scale databases. However this simpler complexity can also result in a loss of accuracy compared to non-linear kernels (e.g. polynomial, RBF, ...).

Recent work exhibits new algorithms scaling linearly in time with the number of training examples. SVMPerf [Joachims, 2006] is a simple *cutting-plane* algorithm for training linear SVMs that is shown to converge in linear time for classification. It is based on SVMstruct, an alternative formulation of the SVM optimization problem originally designed for predicting structured outputs (presented in the next section), that exhibits a different form of sparsity compared to the conventional formulation. The algorithm is empirically very fast and has an intuitively meaningful stopping criterion. *Bundle methods* [Smola *et al.*, 2008] perform in a similar way. LibLinear [Hsieh *et al.*, 2008] also reaches good performance on large scale data sets. Employing an efficient dual coordinate descent procedure, it converges in linear time. Special care has been taken to its implementation as described in [Fan *et al.*, 2008]. As a result, experiments show that LibLinear outperforms SVMPerf in practice.

[2] We also use these notations when we consider SVM problems applied to streams of examples $(x_i, y_i)_{i\geq 1}$.

Solving linear SVMs in the primal can also be very efficient. Recent work on Stochastic Gradient Descent by [Bottou and Bousquet, 2008] have demonstrated that they usually obtain the best generalization performances. For instance, algorithms such as PEGASOS [Shalev-Shwartz *et al.*, 2007] or SVMSGD [Bottou, 2007] are known to be fast and highly scalable online learning solvers. Chapter 3 is entirely devoted to the study of linear SVMs learning. In particular, we discuss in detail how to speed-up Stochastic Gradient Descent and compare empirically SVMSGD and LibLinear.

Most of the methods cited in this section present strong theoretical scaling properties and perform very well for learning SVMs with linear kernels. However one must remember that the picture changes a lot with non-linear kernels because the parameter vector w can no longer be made explicit. Hence, in this case, learning algorithms of the previous sections remain much more efficient.

2.2 SVMs for Structured Output Prediction

This section describes the partial ranking formulation of multiclass SVMs [Crammer and Singer, 2001]. Remarking that structured output prediction is similar to multiclass classification with a very large number of classes, [Tsochantaridis *et al.*, 2005] nicely extend it to deal with all sorts of structures. The presentation first follows their work and then introduces a new parametrization of the dual program.

In the structured setting inputs and outputs to be predicted are more complex than for binary classification. In sequence labeling for example, an input is a sequence of vectors and its output a sequence of atomic class labels. To avoid confusions with the previous section, we now use the following notations: an input pattern is denoted $p \in \mathcal{P}$ and an output is denoted $c \in \mathcal{C}$.

2.2.1 SVM Formulation

As for binary classification, we want to learn a function f that maps patterns $p \in \mathcal{P}$ to outputs $c \in \mathcal{C}$. Patterns can be speech utterances, text sentences, protein sequences, handwritten scans,... Corresponding structured labels can be: speech transcription sequences, grammar parse trees, protein alignments,...

From Multiclass Classification to Structured Output Prediction

When using SVMs, structured output prediction is highly related to multiclass classification which is a well-known task in machine learning. The most widely used approaches combine multiple binary classifiers separately trained using either the *one-versus-all* or *one-versus-one* scheme (e.g. [Hsu and Lin, 2002]). Alternative proposals [Weston and Watkins, 1998, Crammer and Singer, 2001] reformulate the large margin problem to directly address the multiclass problem. These algorithms are more expensive because they must simultaneously handle all the support vectors associated with different inter-class boundaries. Unfortunately, rigorous experiments [Hsu and Lin, 2002, Rifkin and Klautau, 2004] suggest that this higher cost does not translate into higher generalization performance.

The picture changes when, instead of predicting an atomic class label for each input pattern, one targets to produce complex discrete outputs such as sequences, trees, or graphs. Such problems can still be viewed as multiclass (potential outputs can be enumerated, in theory) but with a number of classes growing exponentially with the characteristic size of the output. Yet, dealing with so many classes in a large margin classifier is infeasible without smart factorizations that leverage the specific structure of the outputs (e.g. Section 2.2 or [Taskar *et al.*, 2005]). This

can only be achieved using a direct multiclass formulation because the factorization of the output space implies that all the classes must be handled simultaneously.

Inference

We introduce a discriminant function $S(p, c) \in \mathbb{R}$ that measures the correctness of the association between a pattern p and a class label c. The predicted output can be recovered with the following *inference* step

$$f(p) = \arg\max_{c \in \mathcal{C}} S(p, c) \ . \tag{2.17}$$

This inference step, based on an arg max, is crucial in the formalism we present below. Indeed, Equation (2.17) encodes the process that allows to re-construct any output structure using an input and the model parameters.

For standard multiclass classification, the size of the output space $\mathcal{C}$ remains small. The arg max is simply an exhaustive search. But for compound structures, the size of $\mathcal{C}$ increases and this becomes intractable. One must use the output structure to be able to solve equation (2.17). Modeling dependencies within the output or making conditional-independence assumptions are some common levers. Examples of standard inference procedures can be *Viterbi decoding* for sequences or *Belief-Propagation* for graphs.

All the following formulation is similar to a simple multiclass problem. It becomes valid for any kind of structure as soon as an associated inference process can be modeled within a single arg max equation.

Partial Ranking

We follow here the direct formulation of [Crammer and Singer, 2001] for multiclass classification, and its continuation for large-margin learning with interdependent output spaces by [Altun *et al.*, 2003, Tsochantaridis *et al.*, 2005]. Thus, we assume that the discriminant function has the linear form $S(p, c) = \langle w, \Phi(p, c) \rangle$, where $\Phi(p, c)$ maps the pair (p, c) into a suitable feature space endowed with the dot product $\langle \cdot, \cdot \rangle$.

Consider training patterns $p_1 \dots p_n \in \mathcal{P}$ and their desired outputs $c_1 \dots c_n \in \mathcal{C}$. For each pattern p_i, we want to make sure that the score $S(p_i, c_i)$ of the correct association is greater than the scores $S(p_i, c)$, $c \neq c_i$, of the incorrect associations. This amounts to enforcing a partial order relationship on the elements of $\mathcal{P} \times \mathcal{C}$. This *partial ranking* can be expressed by constraints

$$\forall i = 1 \dots n \quad \forall c \neq c_i \quad \langle w, \delta\Phi_i(c) \rangle \geq \Delta(c_i, c)$$

where $\delta\Phi_i(\bar{c})$ stands for $\Phi(p_i, c_i) - \Phi(p_i, \bar{c})$ and $\Delta(c_i, c)$ is the true loss incurred by predicting label c instead of the true c_i.

Following the standard SVM derivation, [Tsochantaridis *et al.*, 2005] introduce slack variables ξ_i to account for the potential violation of the constraints and optimize a combination of the norm of w and of the size of the slack variables.

$$\min_{w} \quad \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \tag{2.18}$$

$$\text{subject to } \begin{cases} \forall i \quad \xi_i \geq 0 \\ \forall i \quad \forall c \neq c_i \quad \langle w, \delta\Phi_i(c) \rangle \geq \Delta(c_i, c) - \xi_i \end{cases}$$

Dual Programs

The usual derivation leads to solving the following equivalent dual problem (e.g. [Crammer and Singer, 2001, Tsochantaridis *et al.*, 2005]):

$$\begin{aligned} \max_{\boldsymbol{\alpha}} & \sum_{i,c\neq c_i} \Delta(c_i,c)\alpha_i^c - \frac{1}{2}\sum_{\substack{i,c\neq c_i\\ j,\bar{c}\neq c_j}} \alpha_i^c\alpha_j^{\bar{c}}\left\langle \delta\Phi_i(c), \delta\Phi_j(\bar{c})\right\rangle \\ \text{subject to } & \begin{cases} \forall i \quad \forall c\neq c_i \quad \alpha_i^c \geq 0 \\ \forall i \quad \displaystyle\sum_{c\neq c_i} \alpha_i^c \leq C \end{cases} \end{aligned} \tag{2.19}$$

This problem has $n(|\mathcal{C}|-1)$ variables α_i^c, $c\neq c_i$ corresponding to the constraints of (2.18). Once we have the solution, the discriminant function is

$$S(p,c) = \sum_{i,\bar{c}\neq c_i} \alpha_i^{\bar{c}}\left\langle \delta\Phi_i(\bar{c}), \Phi(p,c)\right\rangle$$

This dual problem can be considerably simplified by reparametrizing it with $n|\mathcal{C}|$ variables β_i^c defined as

$$\beta_i^c = \begin{cases} -\alpha_i^c & \text{if } c\neq c_i \\ \displaystyle\sum_{c\neq c_i} \alpha_i^c & \text{otherwise} \end{cases} \tag{2.20}$$

Note that only the $\beta_i^{c_i}$ can be positive. Substituting in (2.19), and taking into account the relation $\sum_c \beta_i^c = 0$, leads to a much simpler expression for the dual problem (the $\delta\Phi_i(\dots)$ have disappeared.)

$$\begin{aligned} \max_{\boldsymbol{\beta}} & -\sum_{i,c} \Delta(c,c_i)\beta_i^c - \frac{1}{2}\sum_{i,j,c,\bar{c}} \beta_i^c\beta_j^{\bar{c}}\left\langle \Phi(p_i,c), \Phi(p_j,\bar{c})\right\rangle \\ \text{subject to } & \begin{cases} \forall i \quad \forall c \quad \beta_i^c \leq \delta(c,c_i)\, C \\ \forall i \quad \displaystyle\sum_{c} \beta_i^c = 0 \end{cases} \end{aligned} \tag{2.21}$$

where $\delta(c,\bar{c})$ is 1 when $c=\bar{c}$ and 0 otherwise. The discriminant function then becomes

$$S(p,c) = \sum_{i,\bar{c}} \beta_i^{\bar{c}}\left\langle \Phi(p_i,\bar{c}), \Phi(p,c)\right\rangle .$$

As usual with kernel machines, the feature mapping function Φ can be defined by the specification of a *joint kernel* function

$$K(p,c,\bar{p},\bar{c}) = \left\langle \Phi(p,c), \Phi(\bar{p},\bar{c})\right\rangle . \tag{2.22}$$

The prediction function is finally rewritten as

$$f(p) = \arg\max_{c\in\mathcal{C}} \sum_{i,\bar{c}} \beta_i^{\bar{c}} K(p_i,\bar{c},p,c). \tag{2.23}$$

Both primal (2.18) and dual (2.21) are very similar to those of standard binary SVMs. However, in this case, computational bottlenecks are (*i*) the size of the constraints set (that might be exponential) and (*ii*) the inference procedure (i.e. the arg max (2.23), that might be costly). Hence, algorithms targeting to tackle structured output prediction must be wise in their ways to crawl the output space and thrifty in arg max computations.

2.2.2 Batch Structured Output Solvers

Batch methods solve the Quadratic Program (2.21) (or (2.19)) with an iterative procedure that run several times over the entire data set until some convergence criterion is met (e.g. [Altun *et al.*, 2003, Tsochantaridis *et al.*, 2005, Taskar *et al.*, 2004, Collins *et al.*, 2008]).

MCSVM The dual cost (2.21) can be seen as a function of a $n \times |\mathcal{C}|$ matrix of Lagrange coefficients where n is the number of examples and $|\mathcal{C}|$ the number of classes. Each iteration of the MCSVM algorithm [Crammer and Singer, 2001] maximizes the restriction of the dual cost to a single row of this coefficient matrix. Successive rows are selected using the gradient of the cost function. That makes MCSVM a very efficient solver of dual (2.21). However, unlike the coefficients matrix, the gradient is not sparse. As a consequence, this approach is not feasible when the number of classes $|\mathcal{C}|$ grows exponentially, because the gradient becomes too large. MCSVM cannot be used to learn generic structured outputs predictors and is restricted to multiclass classification. Yet we use MCSVM as reference in Section 5.2.

Algorithm 5 SVMstruct (ϵ)

1: $\mathcal{S} \leftarrow \emptyset$.
2: **repeat**
3: Pick a random example (p_t, c_t)
4: Set $H(c) = \Delta(c_t, c) - \sum_{(i,\bar{c})\in\mathcal{S}} \beta_i^{\bar{c}} \left(K(p_i, \bar{c}, p_t, c_t) - K(p_i, \bar{c}, p_t, c)\right)$
5: Compute $\hat{c} = \arg\max_{c\in\mathcal{C}} H(c)$
6: Compute $\xi_t = \max\left(0, \max_{\bar{c}\in\mathcal{C} \text{ s.t. } (p_t,\bar{c})\in\mathcal{S}} H(\bar{c})\right)$
7: **if** $H(\hat{c}) \geq \xi_t + \epsilon$ **then**
8: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(t, c_t), (t, \hat{c})\}$
9: Optimize on the set $\mathcal{S}$
10: **end if**
11: Return to step 3.
12: **until** $\mathcal{S}$ has not changed during iteration

SVMstruct Throughout this thesis we use SVMstruct [Tsochantaridis *et al.*, 2005] as batch learning reference. Unlike MCSVM, SVMstruct needs not the full gradient information. It solves the dual problem (2.19) with the clever *cutting plane* algorithm. This ensures convergence while only requiring to store and compute a small fraction of the $n(|\mathcal{C}| - 1)$ constraints as they are added incrementally during training. This point makes SVMstruct suitable for structured output problems with a large number of classes. We display in Algorithm 5 our adaptation of SVMstruct to solve problem (2.21) (with only minor changes compared to the original version of [Tsochantaridis *et al.*, 2005]). At each round a training example is picked in the training set (line 3) and a label corresponding to the input pattern is predicted (line 5). If this prediction violates the constraints set (line 7), it is added to the working set (if not already in). A global optimization step (line 9) is then performed on the constraints set. SVMstruct loops over the whole training set until no more constraints can be added: a theoretical proof ensures that this condition is satisfied in a finite number of optimization steps.

SVMstruct requires an arg max each time a training instance is visited: this strategy allows the cutting plane algorithm to keep a reasonable size for the active constraints set. Nevertheless, combined with the batch mode that iterates several times over the data, this causes the total number of arg max needed by SVMstruct to be much larger than the training set size. As a result, as soon as the output structure gets too sophisticated (e.g. a tree), each arg max becomes

computationally expensive and SVMstruct can only tolerate a small number of training instances for tractability reasons.

Another family of max-margin batch methods is based on the different strategy of output space factorization (e.g. [Taskar *et al.*, 2004]). They solve an alternative problem using additional variables that encode the output structure to ease the computation of the arg max. However, for each example the number of such variables to be added is polynomial in the characteristic size of the outputs, and causes the computational cost of such methods to also grow much more than linearly with the number of examples. Hence, these are impracticable on large data sets.

2.2.3 Online Learning for Structured Outputs

As for binary classification, online methods are scalable alternatives to batch algorithms. As they run a single pass on the training set and update their parameters after each single example (e.g. [Collins, 2002, Daumé III and Marcu, 2005]), their computational cost depends linearly on the number of observations. In particular, the number of inference steps to be performed in the training phase is linear.

Algorithm 6 Structured Perceptron

1: $\mathcal{S} \leftarrow \emptyset$.
2: Pick a random example (p_t, c_t)
3: Compute $f(p_t) = \arg\max_{c \in \mathcal{C}} \sum_{(i,\bar{c}) \in \mathcal{S}} \beta_i^{\bar{c}} K(p_i, \bar{c}, p_t, c)$
4: **if** $f(p) \neq c_t$ **then**
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{(t, c_t), (t, f(p))\}, \quad \beta_t^{c_t} \leftarrow +1, \quad \beta_t^{f(p_t)} \leftarrow -1$
6: **end if**
7: Return to step 2.

Online algorithms inspired by the perceptron [Collins, 2002] can be interpreted as the successive solution of optimization subproblems restricted to coefficients associated with the current training example. Algorithm 6 presents a structured perceptron. Given a training example, it selects the predicted output using an arg max procedure (line 3) but, unlike SVMstruct, it optimizes only on this example (line 5). The random ordering of the training examples drives the successive optimizations. Perceptrons provide strong theoretical guarantees [Graepel *et al.*, 2000] and run very quickly. As for the binary case, large-margin adaptations like passive-aggressive algorithms [Crammer *et al.*, 2006] (which optimize a cost similar to (2.21)), have also been proposed.

2.3 Summary

SVMs are powerful but their training can be problematic in some cases. This chapter does not try to be exhaustive because the number of training methods for SVMs is very large and evolving constantly. However we tried to exhibit the main learning alternatives. In particular we discussed the issues of choosing either an online or a batch algorithm.

It appears that proponents of online algorithms often mention that their generalization bounds are no worse than generalization bounds for batch algorithms [Cesa-Bianchi *et al.*, 2004]. However, the error bounds are not tight and such theoretical guarantees are thus not very informative. Therefore, online algorithms are still significantly less accurate than batch algorithms, as it is confirmed by experimental results displayed in Chapter 5.

In the next chapters, we attempt to fill the gap between online and batch methods by proposing new algorithms for training SVMs scaling like online methods but generalizing like exact ones.

3

Efficient Learning of Linear SVMs with Stochastic Gradient Descent

Contents

When large scale training sets are involved, *Stochastic Gradient Descent* (SGD) algorithms are usually one of the best ways to take advantage of all the data. Indeed, when the bottleneck is the computing time rather than the number of training examples, [Bottou and Bousquet, 2008] established that SGD often yields the best generalization performances, in spite of being poor optimizers.

Nowadays, a growing interest concerns efficient large scale methods. Needless to say, SGD algorithms have been the object of a number of recent works, in particular for training linear SVMs. [Bottou, 2007] and [Shalev-Shwartz *et al.*, 2007] demonstrate that the plain Stochastic Gradient Descent yields particularly effective algorithms when the input patterns are very sparse. It can greatly outperform sophisticated batch methods on large data sets but can also suffer from slow convergence rates especially on ill-conditioned problems. Various remedies have been proposed:

- *Stochastic Meta-Descent* [Schraudolph, 1999] heuristically determines a learning rate for each coefficient of the parameter vector. Although it can solve some ill-conditioning issues, it does not help much for linear SVMs.
- *Natural Gradient Descent* [Amari *et al.*, 2000] replaces the learning rate by the inverse of the Riemannian metric tensor. This quasi-Newton stochastic method is statistically efficient but is penalized in practice by the cost of storing and manipulating the tensor.
- *Online BFGS* (oBFGS) and *Online Limited storage BFGS* (oLBFGS) [Schraudolph *et al.*, 2007] are stochastic adaptations of the Broyden-Fletcher-Goldfarb-Shanno (BFGS)

optimization algorithm. The limited storage version of this algorithm is a quasi-Newton stochastic method whose cost by iteration is a small multiple of the cost of a standard SGD iteration. Unfortunately this penalty is often bigger than the gains associated with the quasi-Newton update.

- *Online Dual Solver* LibLinear [Hsieh *et al.*, 2008] has shown good performance on large scale data sets. These solvers can be applied to both linear and nonlinear SVMs. In the linear case, it is surprisingly close to SGD.

In this chapter we try to identify and leverage different ways to increase SGD abilities to perform well on large scale problems. In particular, we discuss both algorithmic and implementation issues as they are inseparable in this case. This leads us to introduce a new algorithm named SGD-QN, which is a carefully designed Stochastic Gradient Descent for linear Support Vector Machines. SGD-QN won the first PASCAL Large Scale Challenge [Sonnenburg *et al.*, 2008].

Section 3.1.1 presents SGD algorithms for Linear SVMs and analyses the potential gains of quasi-Newton techniques. Sections 3.1.2 and 3.1.3 discuss the sparsity and implementation issues. Finally section 3.2 presents the novel SGD-QN algorithm, and section 3.2.3 reports experimental results. The work presented in this chapter has been the object of a publication (e.g. [Bordes *et al.*, 2009]).

3.1 Stochastic Gradient Descent

This section introduces SGD algorithms and summarizes theoretical results that are relevant to the design of a fast variant of stochastic gradient algorithms. It also exhibits other directions able to improve efficiency.

3.1.1 Analysis

We consider a binary classification problem with training examples $(x, y) \in \mathbb{R}^d \times \{-1, +1\}$. The linear SVM classifier is obtained by minimizing the primal cost function

$$P_n(w) \;=\; \frac{\lambda}{2}\,\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(y_i\,\langle w, x_i\rangle) \;=\; \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\lambda}{2}\,\|w\|^2 + \ell(y_i\,\langle w, x_i\rangle)\right), \tag{3.1}$$

where the hyper-parameter $\lambda > 0$ controls the strength of the regularization term. This formulation is equivalent to the general SVM formulation (2.15) restricted to the set of the n examples and presented in Chapter 2, but using the λ regularization parameter instead of C,[1] a generic loss function ℓ and no bias term. Although typical SVMs could even use non regular convex loss functions, we assume here that the loss $\ell(s)$ is convex and twice differentiable with continuous derivatives ($\ell \in C^2[\mathbb{R}]$). This could be simply achieved by smoothing the traditional loss functions in the vicinity of their non regular points.

Each iteration of the SGD algorithm consists of drawing a random training example (x_t, y_t) and computing a new value of the parameter w_t as

$$w_{t+1} \;=\; w_t - \frac{1}{t+t_0}\,\mathbf{B}\,\boldsymbol{g}_t(w_t) \quad \text{with} \quad \boldsymbol{g}_t(w_t) = \lambda w_t + \ell'(y_t\,\langle w_t, x_t\rangle)\,y_t\,x_t \tag{3.2}$$

and where the *rescaling matrix* $\mathbf{B}$ is positive definite. Since the SVM theory provides simple bounds on the norm of the optimal parameter vector [Shalev-Shwartz *et al.*, 2007], the positive

[1] Corresponding C value is $1/n\lambda$.

constant t_0 is heuristically chosen to ensure that the first few updates do not produce a parameter with an implausibly large norm.

- The traditional first order SGD algorithm, with decreasing learning rate, is obtained by setting $\mathbf{B} = \lambda^{-1}\,\mathbf{I}$ in the generic update (3.2):

$$w_{t+1} \;=\; w_t - \frac{1}{\lambda(t+t_0)}\;\boldsymbol{g}_t(w_t)\,. \tag{3.3}$$

- The second order SGD algorithm is obtained by setting $\mathbf{B}$ to the inverse of the Hessian Matrix $\mathbf{H} = [\,P_n''(w_n^*)\,]$ computed at the optimum w_n^* of the primal cost $P_n(w)$:

$$w_{t+1} \;=\; w_t - \frac{1}{t+t_0}\,\mathbf{H}^{-1}\,\boldsymbol{g}_t(w_t)\,. \tag{3.4}$$

Randomly picking examples could lead to expensive random accesses to the slow memory. In practice, one simply performs sequential passes over the randomly shuffled training set.

What Matters are the Constant Factors

[Bottou and Bousquet, 2008] characterize the asymptotic learning properties of stochastic gradient algorithms in the *large scale regime*, that is, when the bottleneck is the computing time rather than the number of training examples.

Stochastic Gradient Algorithm	**Cost of one iteration**	**Iterations to reach ρ**	**Time to reach accuracy $\boldsymbol{\rho}$**	**Time to reach $\boldsymbol{\mathcal{E} \le c\,(\mathcal{E}_{\text{app}} + \varepsilon)}$**
1^{st} Order SGD	$\mathcal{O}(d)$	$\frac{\nu\kappa^2}{\rho} + \mathrm{o}\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d\nu\kappa^2}{\rho}\right)$	$\mathcal{O}\left(\frac{d\,\nu\,\kappa^2}{\varepsilon}\right)$
2^{nd} Order SGD	$\mathcal{O}(d^2)$	$\frac{\nu}{\rho} + \mathrm{o}\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\nu}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2\,\nu}{\varepsilon}\right)$

Table 3.1: **Asymptotic results for stochastic gradient algorithms.** Reproduced from [Bottou and Bousquet, 2008]. Compare the second last column (time to optimize) with the last column (time to reach the excess test error ϵ). *Legend*: n number of examples; d parameter dimension; c positive constant that appears in the generalization bounds; κ condition number of the Hessian matrix $\mathbf{H}$; $\nu = \mathbf{tr}\left(\mathbf{G}\mathbf{H}^{-1}\right)$ with $\mathbf{G}$ the Fisher matrix (see Theorem 1 for more details). The implicit proportionality coefficients in notations $\mathcal{O}()$ and o() are of course independent of these quantities.

The first three columns of Table 3.1 report the time for a single iteration, the number of iterations needed to reach a predefined accuracy ρ, and their product, the time needed to reach accuracy ρ.

The excess test error $\mathcal{E}$ measures how much the test error is worse than the best possible error for this problem. [Bottou and Bousquet, 2008] decompose the test error as the sum of three terms $\mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}}$. The *approximation error* $\mathcal{E}_{\text{app}}$ measures how closely the chosen family of functions can approximate the optimal solution, The *estimation error* $\mathcal{E}_{\text{est}}$ measures the effect of minimizing the empirical risk instead of the expected risk, The *optimization error* $\mathcal{E}_{\text{opt}}$ measures the impact of the approximate optimization on the generalization performance.

The fourth column of Table 3.1 gives the time necessary to reduce the excess test error $\mathcal{E}$ below a target that depends on $\epsilon > 0$. This is the important metric because the test error is the measure that matters in machine learning.

Both the first order and the second order SGD require a time inversely proportional to ϵ to reach the target test error. Only the constants differ. The second order algorithm is insensitive to the condition number κ of the Hessian matrix but suffers from a penalty proportional to the dimension d of the parameter vector.[2] Therefore, algorithmic changes that exploit the second order information in SGD algorithms are unlikely to yield superlinear speedups. We can at best improve the constant factors.

Limited Storage Approximations of Second Order SGD

Since the second order SGD algorithm is penalized by the high cost of performing the update (3.2) using a full rescaling matrix $\mathbf{B} = \mathbf{H}^{-1}$, it is tempting to consider matrices that admit a sparse representation and yet approximate the inverse Hessian well enough to reduce the negative impact of the condition number κ. The following result precisely describes how the convergence speed of the generic SGD algorithm (3.2) is related to the spectrum of matrix $\mathbf{HB}$.

Theorem 1 *Let $\mathbb{E}_\sigma$ denote the expectation with respect to the random selection of the examples (x_t, y_t) drawn independently from the training set at each iteration. Let $w_n^* = \arg\min_w P_n(w)$ be an optimum of the primal cost. Define the Hessian matrix $\mathbf{H} = \partial^2 P_n(w_n^*)/\partial w^2$ and the Fisher matrix $\mathbf{G} = \mathbf{G}_t = \mathbb{E}_\sigma\left[\boldsymbol{g}_t(w_n^*)\,\boldsymbol{g}_t(w_n^*)^\top\right]$. If the eigenvalues of $\mathbf{HB}$ are in range $\lambda_{\max} \geq \lambda_{\min} > 1/2$, the SGD algorithm (3.2) satisfies*

$$\frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\max}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[P_n(w_t) - P_n(w_n^*)\right] \leq \frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\min}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right)\,.$$

The proof is given below. Note that the theorem assumes that the generic SGD algorithm converges. Convergence in the first-order case holds under very mild assumptions (e.g. [Bottou, 1998]). Convergence in the generic SGD case holds because it reduces to the first-order case with the change of variable $w \to \mathbf{B}^{-\frac{1}{2}}\,w$. Convergence also holds under slightly stronger assumptions when the rescaling matrix $\mathbf{B}$ changes over time (e.g. [Driancourt, 1994]).

Proof Define $\mathbf{v}_t = w_t - w_n^*$ and observe that

$$P_n(w_t) - P_n(w_n^*) \;=\; \mathbf{v}_t^\top \mathbf{H} \mathbf{v}_t + \mathrm{o}\left(t^{-2}\right) \;=\; \mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right) + \mathrm{o}\left(t^{-2}\right)$$

Let $\mathbb{E}_{t-1}$ representing the conditional expectation over the choice of the example at iteration $t-1$ given all the choices made during the previous iterations. Recall that

$$\begin{aligned}
\mathbb{E}_{t-1}\left[\boldsymbol{g}_{t-1}(w_{t-1})\,\boldsymbol{g}_{t-1}(w_{t-1})^\top\right] &= \mathbb{E}_{t-1}\left[\boldsymbol{g}_{t-1}(w_n^*)\,\boldsymbol{g}_{t-1}(w_n^*)^\top\right] + \mathrm{o}\,(1) \;=\; \mathbf{G} + \mathrm{o}\,(1) \\
\text{and}\quad \mathbb{E}_{t-1}\left[\boldsymbol{g}_{t-1}(w_{t-1})\right] &= P_n'(w_{t-1}) \;=\; \mathbf{H}\mathbf{v}_{t-1} + \mathrm{o}\,(\mathbf{v}_{t-1}) \;=\; \mathbf{I}_\varepsilon \mathbf{H}\mathbf{v}_{t-1}
\end{aligned}$$

where notation $\mathbf{I}_\varepsilon$ is a shorthand for $\mathbf{I} + \mathrm{o}\,(1)$, that is, a matrix that converges to the identity. Using the generic SGD update (3.2),

$$\begin{aligned}
\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top &= \mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top - \tfrac{\mathbf{H}\,\mathbf{v}_{t-1}\,\boldsymbol{g}_{t-1}(w_{t-1})^\top\mathbf{B}}{t+t_0} - \tfrac{\mathbf{H}\,\mathbf{B}\,\boldsymbol{g}_{t-1}(w_{t-1})\,\mathbf{v}_{t-1}^\top}{t+t_0} \\
&\quad + \tfrac{\mathbf{H}\,\mathbf{B}\,\boldsymbol{g}_{t-1}(w_{t-1})\,\boldsymbol{g}_{t-1}(w_{t-1})^\top\mathbf{B}}{(t+t_0)^2} \\
\mathbb{E}_{t-1}\left[\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right] &= \mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top - \tfrac{\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top\,\mathbf{H}\,\mathbf{I}_\varepsilon\,\mathbf{B}}{t+t_0} - \tfrac{\mathbf{H}\,\mathbf{B}\,\mathbf{I}_\varepsilon\,\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top}{t+t_0} + \tfrac{\mathbf{HBGB}}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right) \\
\mathbb{E}_{t-1}\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] &= \mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right) - \tfrac{2\,\mathbf{tr}\left(\mathbf{H}\,\mathbf{B}\,\mathbf{I}_\varepsilon\,\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top\right)}{t+t_0} + \tfrac{\mathbf{tr}(\mathbf{HBGB})}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right) \\
\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] &= \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)\right] - \tfrac{2\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\,\mathbf{B}\,\mathbf{I}_\varepsilon\,\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top\right)\right]}{t+t_0} + \tfrac{\mathbf{tr}(\mathbf{HBGB})}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right)\,.
\end{aligned}$$

[2] [Bottou and Bousquet, 2008] obtain slightly worse scaling laws for non-stochastic gradient algorithms.

Let $\lambda_{\max} \geq \lambda_{\min} > 1/2$ be the extreme eigenvalues of $\mathbf{HB}$. Since, for any positive matrix $\mathcal{X}$,

$$\left(\lambda_{\min} + \mathrm{o}\,(1)\right) \mathbf{tr}\,(\mathcal{X}) \;\leq\; \mathbf{tr}\,(\mathbf{HBI}_\varepsilon \mathcal{X}) \;\leq\; \left(\lambda_{\max} + \mathrm{o}\,(1)\right) \mathbf{tr}\,(\mathcal{X})$$

we can bracket $\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right]$ between the expressions

$$\left(1 - \frac{2\lambda_{\max}}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right) \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top\right)\right] \;+\; \frac{\mathbf{tr}\,(\mathbf{HBGB})}{(t+t_0)^2} \;+\; \mathrm{o}\left(t^{-2}\right)$$

and

$$\left(1 - \frac{2\lambda_{\min}}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right) \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\,\mathbf{v}_{t-1}\,\mathbf{v}_{t-1}^\top\right)\right] \;+\; \frac{\mathbf{tr}\,(\mathbf{HBGB})}{(t+t_0)^2} \;+\; \mathrm{o}\left(t^{-2}\right)$$

By recursively applying this bracket, we obtain

$$u_{\lambda_{\max}}(t+t_0) \;\leq\; \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] \;\leq\; u_{\lambda_{\min}}(t+t_0)$$

where the notation $u_\lambda(t)$ represents a sequence of real satisfying the recursive relation

$$u_\lambda(t) \;=\; \left(1 - \frac{2\lambda}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right) u_\lambda(t-1) + \frac{\mathbf{tr}\,(\mathbf{HBGB})}{t^2} + \mathrm{o}\left(\frac{1}{t^2}\right).$$

From [Bottou and Le Cun, 2005, lemma 1], $\lambda > 1/2$ implies $t\,u_\lambda(t) \longrightarrow \frac{\mathbf{tr}(\mathbf{HBGB})}{2\lambda-1}$. Then

$$\frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\max}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] \leq \frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\min}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right)$$

and

$$\frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\max}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[P_n(w_t) - P_n(w_n^*)\right] \leq \frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\min}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

□

The following two corollaries recover the maximal number of iterations listed in Table 3.1 with $\nu = \mathbf{tr}\left(\mathbf{G}\mathbf{H}^{-1}\right)$ and $\kappa = \lambda^{-1}\|\mathbf{H}\|$.

Corollary 2 *Assume $\mathbf{B} = \mathbf{H}^{-1}$ as in the second order SGD algorithm* (3.4)*. We have then*

$$\mathbb{E}_\sigma\left[P_n(w_t) - P_n(w_n^*)\right] \;=\; \mathbf{tr}\left(\mathbf{G}\mathbf{H}^{-1}\right) t^{-1} + \mathrm{o}\left(t^{-1}\right) \;=\; \nu\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

Corollary 3 *Assume $\mathbf{B} = \lambda^{-1}\,\mathbf{I}$ as in the first order SGD algorithm* (3.3)*. We have then*

$$\mathbb{E}_\sigma\left[P_n(w_t) - P_n(w_n^*)\right] \;\leq\; \lambda^{-2}\,\mathbf{tr}\left(\mathbf{H}^2\mathbf{G}\mathbf{H}^{-1}\right) t^{-1} + \mathrm{o}\left(t^{-1}\right) \;\leq\; \kappa^2\,\nu\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

An often rediscovered property of second order SGD provides an useful reference point:

Theorem 4 ([Fabian, 1973, Murata and Amari, 1999, Bottou and Le Cun, 2005]) *Let $w^* = \arg\min \frac{\lambda}{2}\|w\|^2 + \mathbb{E}_{x,y}\left[\,\ell(y\,\langle w, x\rangle)\,\right]$. Given a sample of n independent examples (x_i, y_i), define $w_n^* = \arg\min_w P_n(w)$ and compute w_n by applying the second order SGD update* (3.4) *to each of the n examples. Then both $n\,\mathbb{E}\left[\,\|w_n - w^*\|^2\right]$ and $n\,\mathbb{E}\left[\,\|w_n^* - w^*\|^2\right]$ converge to a same positive constant K when n increases.*

This result means that, asymptotically and on average, the parameter w_n obtained after one pass of second order SGD is as close to the infinite training set solution w^* as the true optimum of the primal w_n^*. Therefore, when the training set is large enough, we can expect that a single pass of second order SGD is sufficient to replicate the test error of the actual SVM solution.

When we replace the full second order rescaling matrix $\mathbf{B} = \mathbf{H}^{-1}$ by a computationally more acceptable approximation, Theorem 1 indicates that we lose a constant factor on the required number of iterations. We need to perform several passes over the randomly reshuffled training set. On the other hand, a well chosen approximation of the rescaling matrix can save a large constant factor on the computation of the generic SGD update (3.2).

The best training times are therefore obtained by carefully trading the quality of the approximation for sparse representations.

	Frequency	**Loss**
Special example:	$\frac{n}{\texttt{skip}}$	$\frac{\lambda\,\texttt{skip}}{2}\,\|w\|^2$
Examples 1 to n:	1	$\ell(y_i w^\top x_i)$

Table 3.2: **Frequencies and losses.** The regularization term in the primal cost can be viewed as an additional training example with an arbitrarily chosen frequency and a specific loss function.

More Speedup Opportunities

We have argued that carefully designed quasi-Newton techniques can save a constant factor on the training times. There are of course many other ways to save constant factor:

- *Exploiting the sparsity of the patterns* (see Section 3.1.2) can save a constant factor in the cost of each iteration. However their benefit is more limited in the second-order case, because the inverse Hessian matrix is not sparse.
- *Implementation details* (see Section 3.1.3) such as compiler technology or parallelization can also reduce the learning time by constant factors.

Such opportunities are often dismissed as engineering tricks. However they should be considered on an equal footing with quasi-Newton techniques. Constant factors matter regardless of their origin. The following two sections provide a detailed discussion of sparsity and implementation.

3.1.2 Scheduling Stochastic Updates to Exploit Sparsity

First order SGD iterations can be made substantially faster when the patterns x_t are sparse. The first order SGD update has the form

$$w_{t+1} \;=\; w_t - \alpha_t w_t - \beta_t x_t\,, \tag{3.5}$$

where α_t and β_t are scalar coefficients. Subtracting $\beta_t x_t$ from the parameter vector involves solely the nonzero coefficients of the pattern x_t. On the other hand, subtracting $\alpha_t w_t$ involves all d coefficients. A naive implementation of (3.5) would therefore spend most of the time processing this first term. [Shalev-Shwartz *et al.*, 2007] circumvent this problem by representing the parameter w_t as the product $s_t \mathbf{v}_t$ of a scalar and a vector. The update (3.5) can then be computed as $s_{t+1} = (1-\alpha_t)s_t$ and $\mathbf{v}_{t+1} = \mathbf{v}_t - \beta x_t / s_{t+1}$ in time proportional to the number of nonzero coefficients in x_t.

Although this simple approach works well for the first order SGD algorithm, it does not extend nicely to quasi-Newton SGD algorithms. A more general method consists of treating the regularization term in the primal cost (3.1) as an additional training example occurring with an arbitrarily chosen frequency with a specific loss function.

Consider examples with the frequencies and losses listed in table 3.2 and write the average loss:

$$\frac{1}{\frac{n}{\texttt{skip}}+n}\left[\frac{n}{\texttt{skip}}\left(\frac{\lambda\,\texttt{skip}}{2}\|w\|^2\right) \;+\; \sum_{i=1}^{n}\ell(y_i\;\langle w,x_i\rangle)\right] = \frac{\texttt{skip}}{1+\texttt{skip}}\left[\frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\ell(y_i\,\langle w,x_i\rangle)\right].$$

Minimizing this loss is of course equivalent to minimizing the primal cost (3.1) with its regularization term. Applying the SGD algorithm to the examples defined in table 3.2 separates

Algorithm 7 Comparison of the pseudo-codes of SGD and SVMSGD2.

SGD	SVMSGD2
Require: λ, w_0, t_0, T	**Require:** λ, w_0, t_0, T, `skip`
1: $t = 0$	1: $t = 0$, `count` $=$ `skip`
2: **while** $t \leq T$ **do**	2: **while** $t \leq T$ **do**
3: $w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)}(\lambda w_t + \ell'(y_t \langle w_t, x_t \rangle) y_t x_t)$	3: $w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)}\ell'(y_t \langle w_t, x_t \rangle) y_t x_t$
4:	4: `count` $=$ `count`-1
5:	5: **if** `count` < 0 **then**
6:	6: $w_{t+1} = w_{t+1} - \frac{\texttt{skip}}{t+t_0} w_{t+1}$
7:	7: `count` $=$ `skip`
8:	8: **end if**
9: $t = t + 1$	9: $t = t + 1$
10: **end while**	10: **end while**
11: **return** w_T	11: **return** w_T

the regularization updates, which involve the special example, from the pattern updates, which involve the real examples. The parameter `skip` regulates the relative frequencies of these updates. The SVMSGD2 algorithm [Bottou, 2007] measures the average pattern sparsity and picks a frequency that ensures that the amortized cost of the regularization update is proportional to the number of nonzero coefficients. Algorithm 7 compares the pseudo-codes of the naive first order SGD and of the first order SVMSGD2. Both algorithms handle the real examples at each iteration (line 3) but SVMSGD2 only performs a regularization update every `skip` iterations (line 6).

Assume s is the average proportion of nonzero coefficients in the patterns x_i and set `skip` to c/s where c is a predefined constant (we use $c = 16$ in our experiments). Each pattern update (line 3) requires sd operations. Each regularization update (line 6) requires d operations but occurs s/c times less often. The average cost per iteration is therefore proportional to $\mathcal{O}(sd)$ instead of $\mathcal{O}(d)$.

3.1.3 Implementation

In the optimization literature, a superior algorithm implemented with a slow scripting language usually beats careful implementations of inferior algorithms. This is because the superior algorithm minimizes the training error with a higher order convergence.

This is no longer true in the case of large scale machine learning because we care about the test error instead of the training error. As explained above, algorithm improvements do not improve the order of the test error convergence. They can simply improve constant factors and therefore *compete evenly with implementation improvements*. Time spent refining the implementation is time well spent.

- There are lots of methods for representing sparse vectors with sharply different computing requirement for sequential and random access. Our C++ implementations use either a full vector representation or a sparse vector representation consisting of an ordered list of index/value pairs (see Table 3.3.)

  Our implementation always uses a full vector for the parameter w and picks a format for the patterns x according to the average sparsity of the data set. Inappropriate choices cost outrageous time penalties. For example, on a dense data set with 500 attributes, using sparse vectors increases the training time by 50%; on the sparse RCV1 data set (see

	Full	**Sparse**
Random access to a single coefficient:	$\mathcal{O}(1)$	$\mathcal{O}(s)$
In-place addition into a full vector of dimension d:	$\mathcal{O}(d)$	$\mathcal{O}(s)$
In-place addition into a sparse vector with s' nonzeros:	$\mathcal{O}(d+s')$	$\mathcal{O}(s+s')$

Table 3.3: **Costs of various operations** on a vector of dimension d with s nonzero coefficients.

Table 5.4), using a sparse vector to represent the parameter w increases the training time by more than 900%.

- Modern processors often sport specialized instructions to handle vectors and multiple cores. Linear algebra libraries, such as `BLAS`, may or may not use them in ways that suit our purposes. Compilation flags have nontrivial impacts on the learning times.

Such implementation improvements are often (but not always) orthogonal to the algorithmic improvements described above. The main issue consists of deciding how much development resources are allocated to implementation and to algorithm design. This trade-off depends on the available competencies.

3.2 SGD-QN: A Careful Diagonal Quasi-Newton SGD

As explained in Section 3.1.1, designing an efficient quasi-Newton SGD algorithm involves a careful trade-off between the sparsity of the scaling matrix representation $\mathbf{B}$ and the quality of its approximation of the inverse hessian $\mathbf{H}^{-1}$. The two obvious choices are diagonal approximations [Becker and Le Cun, 1989] and low rank approximations [Schraudolph *et al.*, 2007].

3.2.1 Rescaling Matrices

Diagonal Rescaling Matrices

Among numerous practical suggestions for running SGD algorithm in multilayer neural networks, [Le Cun *et al.*, 1998] emphatically recommend to rescale the input space in order to improve the condition number κ of the Hessian matrix. In the case of a linear model, such preconditioning is similar to using a constant diagonal scaling matrix.

Rescaling the input space defines transformed patterns $\mathcal{X}_t$ such that $[\mathcal{X}_t]_i = b_i[x_t]_i$ where the notation $[\mathbf{v}]_i$ represents the i-th coefficient of vector $\mathbf{v}$. This transformation does not change the classification if the parameter vectors are modified as $[\mathbf{W}_t]_i = [w_t]_i / b_i$. The first order SGD update on these modified variable is then

$$\begin{aligned} \forall i = 1 \dots d \qquad [\mathbf{W}_{t+1}]_i &= [\mathbf{W}_t]_i - \eta_t \left(\lambda [\mathbf{W}_t]_i + \ell'(y_t \langle \mathbf{W}_t, \mathcal{X}_t \rangle) y_t [\mathcal{X}_t]_i, \right) \\ &= [\mathbf{W}_t]_i - \eta_t \left(\lambda [\mathbf{W}_t]_i + \ell'(y_t \langle w_t, x_t \rangle) y_t b_i [x_t]_i \right) . \end{aligned}$$

Multiplying by b_i shows how the original parameter vector w_t are affected:

$$\forall i = 1 \dots d \qquad [w_{t+1}]_i = [w_t]_i - \eta_t \left(\lambda [w_t]_i + \ell'(y_t \langle w_t, x_t \rangle) y_t b_i^2 [x_t]_i \right) .$$

We observe that rescaling the input is equivalent to multiplying the gradient by a fixed diagonal matrix $\mathbf{B}$ whose elements are the squares of the coefficients b_i.

Ideally we would like to make the product $\mathbf{BH}$ spectrally close the identity matrix. Unfortunately we do not know the value of the Hessian matrix $\mathbf{H}$ at the optimum w^*. Instead we could consider the current value of the Hessian $\mathbf{H}_{w_t} = P''(w_t)$ and compute the diagonal rescaling matrix $\mathbf{B}$ that makes $\mathbf{BH}_{w_t}$ closest to the identity. This computation could be very costly because it involves the full Hessian matrix. [Becker and Le Cun, 1989] approximate the optimal diagonal rescaling matrix by inverting the diagonal coefficients of the Hessian. The method relies on the analytical derivation of these diagonal coefficients for multilayer neural networks. This derivation does not extend to arbitrary models. It certainly does not work in the case of traditional SVMs because the hinge loss has zero curvature almost everywhere.

Low Rank Rescaling Matrices

The popular LBFGS optimization algorithm [Nocedal, 1980] maintains a low rank approximation of the inverse Hessian by storing the k most recent rank-one BFGS updates instead of the full inverse Hessian matrix. When the successive full gradients $P'_n(w_{t-1})$ and $P'_n(w_t)$ are available, standard rank one updates can be used to directly estimate the inverse Hessian matrix $\mathbf{H}^{-1}$. Using this method with stochastic gradient is tricky because the full gradients $P'_n(w_{t-1})$ and $P'_n(w_t)$ are not readily available. Instead we only have access to the stochastic estimates $\boldsymbol{g}_{t-1}(w_{t-1})$ and $\boldsymbol{g}_t(w_t)$ which are too noisy to compute good rescaling matrices.

The oLBFGS algorithm [Schraudolph *et al.*, 2007] compares instead the derivatives $\boldsymbol{g}_{t-1}(w_{t-1})$ and $\boldsymbol{g}_{t-1}(w_t)$ for the same example (x_{t-1}, y_{t-1}). This reduces the noise to an acceptable level at the expense of the computation of the additional gradient vector $\boldsymbol{g}_{t-1}(w_t)$.

Compared to the first order SGD, each iteration the oLBFGS algorithms computes the additional quantity $\boldsymbol{g}_{t-1}(w_t)$ and updates the list of k rank one updates. The most expensive part however remains the multiplication of the gradient $\boldsymbol{g}_t(w_t)$ by the low-rank estimate of the inverse Hessian. With $k = 10$, each iteration of our oLBFGS implementation runs empirically 11 times slower than a first order SGD iteration.

3.2.2 SGD-QN

The SGD-QN algorithm estimates a *diagonal rescaling matrix* using a technique inspired by oLBFGS. For any pair of parameters w_{t-1} and w_t, a Taylor series of the gradient of the primal cost P provides the secant equation:

$$w_t - w_{t-1} \approx \mathbf{H}_{w_t}^{-1} \left(P'_n(w_t) - P'_n(w_{t-1}) \right). \tag{3.6}$$

We would then like to replace the inverse Hessian matrix $\mathbf{H}_{w_t}^{-1}$ by a diagonal estimate $\mathbf{B}$

$$w_t - w_{t-1} \approx \mathbf{B} \left(P'_n(w_t) - P'_n(w_{t-1}) \right).$$

Since we are designing a stochastic algorithm, we do not have access to the full gradient P'_n. Following oLBFGS, we replace them by the local gradients $\boldsymbol{g}_{t-1}(w_t)$ and $\boldsymbol{g}_{t-1}(w_{t-1})$ and obtain

$$w_t - w_{t-1} \approx \mathbf{B} \left(\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1}) \right).$$

Since we chose to use a diagonal rescaling matrix $\mathbf{B}$, we can write the term-by-term equality

$$[w_t - w_{t-1}]_i \approx \mathbf{B}_{ii} \left[\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1}) \right]_i,$$

where the notation $[\mathbf{v}]_i$ still represents the i-th coefficient of vector $\mathbf{v}$. This leads to computing $\mathbf{B}_{ii}$ as the average of the ratio $[w_t - w_{t-1}]_i / \left[\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1}) \right]_i$. An online estimation is

easily achieved during the course of learning by performing a leaky average of these ratios,

$$\mathbf{B}_{ii} \leftarrow \mathbf{B}_{ii} + \frac{2}{r}\left(\frac{[w_t - w_{t-1}]_i}{\big[\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1})\big]_i} - \mathbf{B}_{ii} \right) \qquad \forall i = 1 \dots d\,, \tag{3.7}$$

and where the integer r is incremented whenever we update the coefficient $\mathbf{B}_{ii}$.

The weights of the scaling matrix $\mathbf{B}$ are initialized to λ^{-1} because this corresponds to the exact setup of first order SGD. Since the curvature of the primal cost (3.1) is always larger than λ, the ratios $\big[\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1})\big]_i / [w_t - w_{t-1}]_i$ are always larger than λ. Therefore the coefficients $\mathbf{B}_{ii}$ never exceed their initial value λ^{-1}. Basically these scaling factors slow down the convergence along some axes. The speedup does not occur because we follow the trajectory faster, but because we follow a better trajectory.

Performing the weight update (3.2) with a diagonal rescaling matrix $\mathbf{B}$ consists in performing term-by-term operations with a time complexity that is marginally greater than the complexity of the first order SGD (3.3) update. The computation of the additional gradient vector $\boldsymbol{g}_{t-1}(w_t)$ and the re-estimation of all the coefficients $\mathbf{B}_{ii}$ essentially triples the computing time of a first order SGD iteration with non-sparse inputs (3.3), and is considerably slower than a first order SGD iteration with sparse inputs implemented as discussed in Section 3.1.2.

Fortunately this higher computational cost per iteration can be nearly avoided by scheduling the re-estimation of the rescaling matrix with the same frequency as the regularization updates. Section 3.2.1 has shown that a diagonal rescaling matrix does little more than rescaling the input variables. Since a fixed diagonal rescaling matrix already works quite well, there is little need to update its coefficients very often.

Algorithm 8 compares the SVMSGD2 and SGD-QN algorithms. Whenever SVMSGD2 performs a regularization update, we set the flag `updateB` to schedule a re-estimation of the rescaling coefficients during the next iteration. This is appropriate because both operations have comparable computing times. Therefore the rescaling matrix re-estimation schedule can be regulated with the same `skip` parameter as the regularization updates. In practice, we observe that each SGD-QN iteration demands less than twice the time of a first order SGD iteration.

Because SGD-QN re-estimates the rescaling matrix after a pattern update, special care must be taken when the ratio $[w_t - w_{t-1}]_i / \big[\boldsymbol{g}_{t-1}(w_t) - \boldsymbol{g}_{t-1}(w_{t-1})\big]_i$ has the form 0/0 because the corresponding input coefficient $[x_{t-1}]_i$ is zero. Since the secant equation (3.6) is valid for any two values of the parameter vector, one computes the ratios with parameter vectors w_{t-1} and $w_t + \boldsymbol{\varepsilon}$ and derives the correct value by continuity. When $[x_{t-1}]_i = 0$, we can write

$$\begin{aligned}
\frac{[(w_t+\boldsymbol{\varepsilon})-w_{t-1}]_i}{\big[\boldsymbol{g}_{t-1}(w_t+\boldsymbol{\varepsilon})-\boldsymbol{g}_{t-1}(w_{t-1})\big]_i} &= \frac{[(w_t+\boldsymbol{\varepsilon})-w_{t-1}]_i}{\lambda[(w_t+\boldsymbol{\varepsilon})-w_{t-1}]_i + \Big(\ell'\big(y_{t-1}\langle(w_t+\boldsymbol{\varepsilon}),x_{t-1}\rangle\big)-\ell'\big(y_{t-1}\langle w_{t-1},x_{t-1}\rangle\big)\Big)\, y_{t-1}\,[x_{t-1}]_i} \\
&= \left(\lambda + \frac{\Big(\ell'\big(y_{t-1}\langle(w_t+\boldsymbol{\varepsilon}),x_{t-1}\rangle\big)-\ell'\big(y_{t-1}\langle w_{t-1},x_{t-1}\rangle\big)\Big)\, y_{t-1}\,[x_{t-1}]_i}{[(w_t+\boldsymbol{\varepsilon})-w_{t-1}]_i}\right)^{-1} \\
&= \left(\lambda + \frac{0}{[\boldsymbol{\varepsilon}]_i}\right)^{-1} \quad \stackrel{\boldsymbol{\varepsilon}\to 0}{\longrightarrow} \quad \lambda^{-1}\,.
\end{aligned}$$

3.2.3 Experiments

We demonstrate the good scaling properties of SGD-QN in two ways: we present a detailed comparison with other stochastic gradient methods, and we summarize the results obtained on the PASCAL Large Scale Challenge.

Table 3.4 describes the three binary classification tasks we used for comparative experiments. The Alpha and Delta tasks were defined for the PASCAL Large Scale Challenge [Sonnenburg

Algorithm 8 Comparison of the pseudo-codes of SVMSGD2 and SGD-QN.

SVMSGD2	SGD-QN
Require: λ, w_0, t_0, T, `skip`	**Require:** λ, w_0, t_0, T, `skip`
$t = 0$, `count` $=$ `skip`	$t = 0$, `count` $=$ `skip`,
	$\mathbf{B} = \lambda^{-1}\mathbf{I}$, `updateB` $=$ `false`, $r = t_0/$`skip`.
while $t \leq T$ **do**	**while** $t \leq T$ **do**
$w_{t+1} = w_t - \frac{1}{\lambda(t+t_0)}\ell'(y_t \langle w_t, x_t\rangle) y_t x_t$	$w_{t+1} = w_t - (t+t_0)^{-1}\ell'(y_t \langle w_t, x_t\rangle) y_t \mathbf{B}\, x_t$
	if `updateB` $=$ `true` **then**
	$\mathbf{p}_t = \boldsymbol{g}_t(w_{t+1}) - \boldsymbol{g}_t(w_t)$
	$\forall i$, $\mathbf{B}_{ii} = \mathbf{B}_{ii} + \frac{2}{r}\left([w_{t+1} - w_t]_i\,[\mathbf{p}_t]_i^{-1} - \mathbf{B}_{ii}\right)$
	$\forall i$, $\mathbf{B}_{ii} = \max(\mathbf{B}_{ii}, 10^{-2}\lambda^{-1})$
	$r = r + 1$, `updateB` $=$ `false`
	end if
`count` $=$ `count`-1	`count` $=$ `count`-1
if `count` < 0 **then**	**if** `count` < 0 **then**
$w_{t+1} = w_{t+1} -$ `skip` $(t+t_0)^{-1} w_{t+1}$	$w_{t+1} = w_{t+1} -$ `skip` $(t+t_0)^{-1}\lambda\,\mathbf{B}\, w_{t+1}$
`count` $=$ `skip`	`count` $=$ `skip`, `updateB` $=$ `true`
end if	**end if**
$t = t + 1$	$t = t + 1$
end while	**end while**
return w_T	**return** w_T

Data set	Train. Ex.	Test. Ex.	Features	s	λ	t_0	*skip*
Alpha	100,000	50,000	500	1	10^{-5}	10^6	16
Delta	100,000	50,000	500	1	10^{-4}	10^4	16
RCV1	781,265	23,149	47,152	0.0016	10^{-4}	10^5	9,965

Table 3.4: **Data sets and parameters used for experiments.**

et al., 2008]. We train with the first 100,000 examples and test with the last 50,000 examples of the official training sets because the official testing sets are not available. Alpha and Delta are dense data sets with relatively severe conditioning problems. The third task is the classification of RCV1 documents belonging to class CCAT [Lewis *et al.*, 2004]. This task has become a standard benchmark for linear SVMs on sparse data. Despite its larger size, the RCV1 task is much easier than the Alpha and Delta tasks. All methods discussed in this paper performs well on RCV1.

The experiments reported in the last paragraph of this section use the hinge loss $\ell(s) = \max(0, 1-s)$. All other experiments use the squared hinge loss $\ell(s) = \frac{1}{2}(\max(0, 1-s))^2$. In practice, there is no need to make the losses twice differentiable by smoothing their behavior near $s = 0$. Unlike most batch optimizer, stochastic algorithms do not aim directly for non-differentiable points, but randomly hop around them. The stochastic noise implicitly smoothes the loss.

The SGD, SVMSGD2, oLBFGS, and SGD-QN algorithms were implemented using the same C++ code base. Implementations and experiment scripts are freely available under the `GNU Public License` as part of the `libsgdqn` library on `http://www.mloss.org` (go to `http://mloss.org/software/view/197/`).

	Alpha	RCV1
SGD	0.13	36.8
SVMSGD2	0.10	0.20
SGD-QN	0.21	0.37

Table 3.5: **Time (sec.) for performing one pass over the training set.**

All experiments are carried out in single precision. We did not experience numerical accuracy issues, probably because of the influence of the regularization term. Our implementation of oLBFGS maintains a rank 10 rescaling matrix. Setting the oLBFGS gain schedule is rather delicate. We obtained fairly good results by replicating the gain schedule of the VieCRF package.[3] We also propose a comparison with the online dual linear SVM solver [Hsieh *et al.*, 2008] implemented in the LibLinear package.[4] We did not re-implement this algorithm because the LibLinear implementation has proved as simple and as efficient as ours.

The t_0 parameter is determined using an automatic procedure: since the size of the training set does not affect results of Theorem 1, we simply pick a subset containing 10% of the training examples, perform one SGD-QN pass over this subset with several values for t_0, and pick the value for which the primal cost decreases the most. These values are given in Table 3.4.

Sparsity Tricks

The influence of the scheduling tricks described in Section 3.1.2 is illustrated in Table 3.5. There are displayed the training times of SGD and SVMSGD2. The latter uses scheduling tricks while SGD does not. SVMSGD2 enjoys shorter training durations, especially with sparse data, where it is more than 180 times faster. This table also demonstrates that an iteration of the quasi-newton SGD-QN is not prohibitively expensive.

Quasi-Newton

Figure 3.1 shows how the primal cost $P_n(w)$ of the Delta data set evolves with the number of passes (left) and the training time (right). Compared to the first order SVMSGD2, both the oLBFGS and SGD-QN algorithms dramatically decrease the number of passes required to achieve similar values of the primal. Even if it uses a more precise approximation of the inverse Hessian, oLBFGS does not perform better after a single pass than SGD-QN. Besides, running a single pass of oLBFGS is much slower than running multiple passes of SVMSGD2 or SGD-QN. The benefits of its second-order approximation are canceled by its greater time requirements per iteration. On the other hand, each SGD-QN iteration is only marginally slower than a SVMSGD2 iteration; the reduction of the number of iterations is sufficient to offset this cost.

Training Speed

Figure 3.2 displays the test errors achieved on the Alpha, Delta and RCV1 data sets as a function of the number of passes (left) and the training time (right). These results show again than both oLBFGS and SGD-QN require less iterations than SVMSGD2 to achieve the same test error. However oLBFGS suffers from the relatively high complexity of its update process. The

[3] `http://www.ofai.at/~jeremy.jancsary`
[4] `http://www.csie.ntu.edu.tw/~cjlin/liblinear`

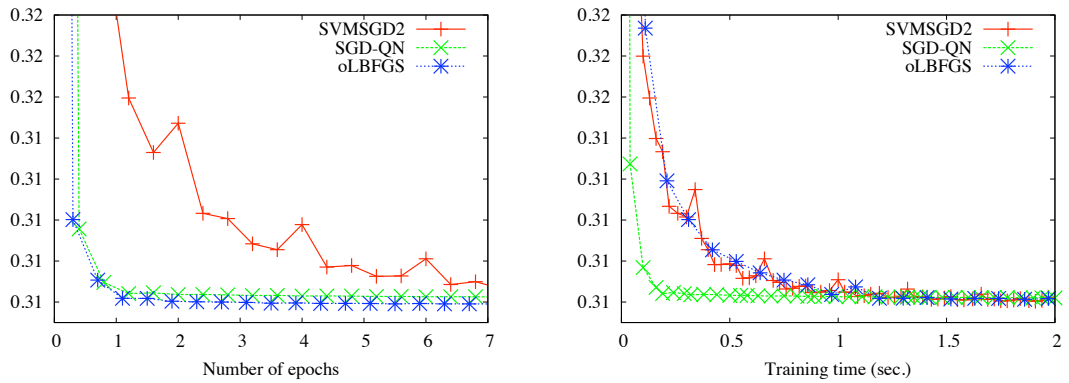


Figure 3.1: **Primal costs** according to the number of epochs (left) and the training duration (right) on the DELTA data set.

SGD-QN algorithm runs significantly faster than the dual solver LibLinear on both the dense data sets ALPHA and DELTA; and the sparse RCV1 data set.

LibLinear automatically computes its learning rate in the dual: this can be seen as an advantage since this removes an extra-parameter to tune. However, our experiments show that, when carefully used, the freedom of choice of a SGD learning rate can lead to faster training.

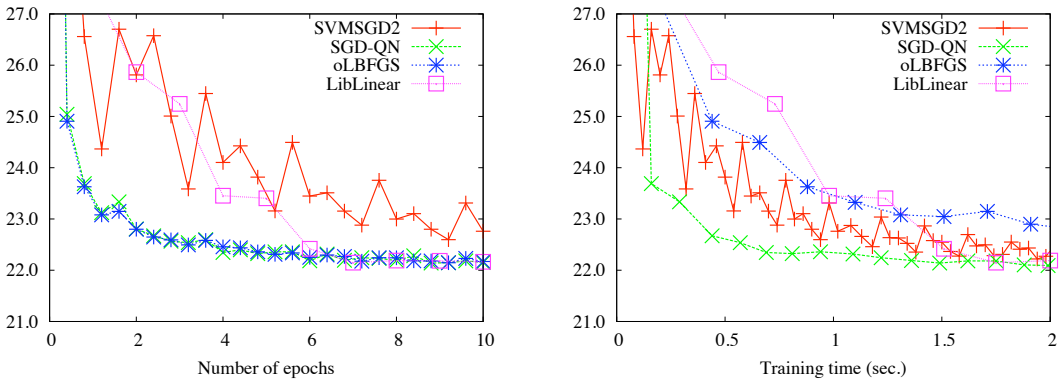
According to Theorem 4, given a large enough training set, a perfect second order SGD algorithm would reach the batch test error after a single pass. One pass learning is attractive when we are dealing with high volume streams of examples that cannot be stored and retrieved quickly. Figure 3.2 (left) shows that both oLBFGS and SGD-QN are close to that ideal (oLBFGS might even be a little closer). They would become even more attractive for problems where the example retrieval time is much greater than the computing time.

PASCAL Large Scale Challenge Results

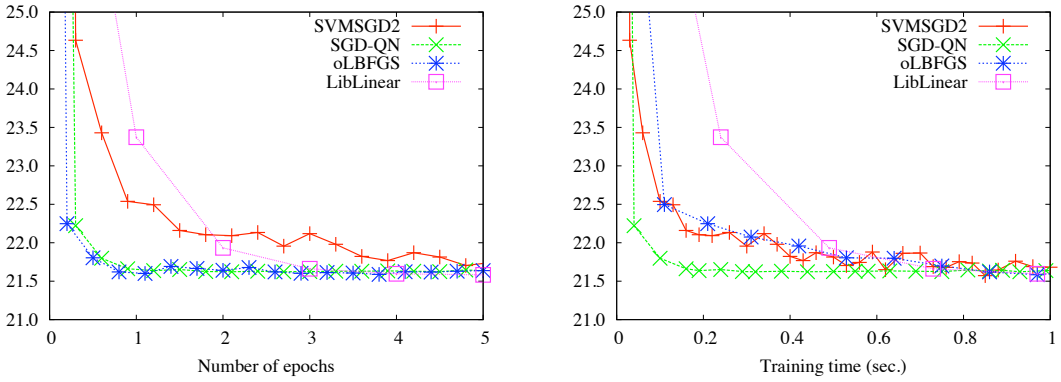
The first PASCAL Large Scale Learning Challenge [Sonnenburg *et al.*, 2008] was designed to identify which machine learning techniques best address these new concerns. A generic evaluation framework and various data sets have been provided. Evaluations were carried out on the basis of various performance curves such as training time versus test error, data set size versus test error, and data set size versus training time[5].

Given its strong generalization and scaling properties, SGD-QN was a natural choice for the "Wild Track" of the competition which focuses on the relation between training time and test performance. Wild track contributors were free to do anything leading to more efficient and more accurate methods. Forty two methods have been submitted to this track. Table 3.6 shows the SGD-QN ranks determined by the organizers of the challenge according to their evaluation criteria. The SGD-QN algorithm always ranks among the top five submissions and ranks first in overall score (tie with another Newton method).

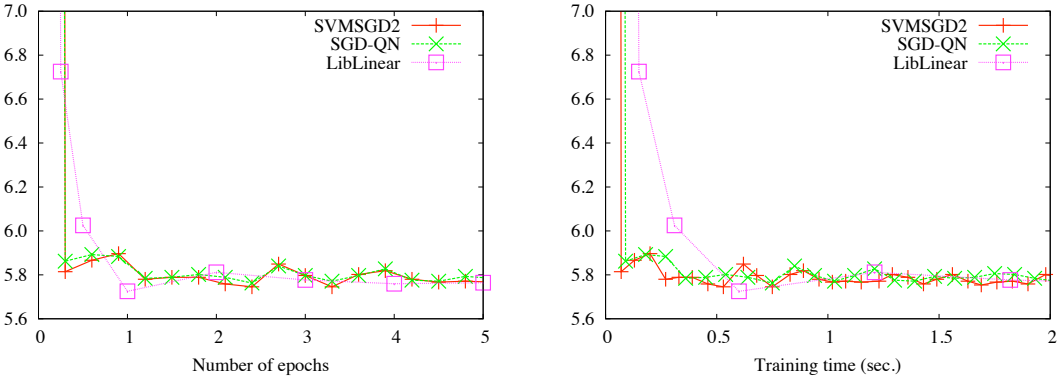
[5]This material and its documentation can be found at `http://largescale.first.fraunhofer.de/`



ALPHA DATA SET



DELTA DATA SET



RCV1 DATA SET

Figure 3.2: **Test errors (in %)** according to the number of epochs (left) and training duration (right).

Data set	λ	*skip*	Passes	**Rank**
Alpha	10^{-5}	16	10	$\mathbf{1}^{st}$
Beta	10^{-4}	16	15	$\mathbf{3}^{rd}$
Gamma	10^{-3}	16	10	$\mathbf{1}^{st}$
Delta	10^{-3}	16	10	$\mathbf{1}^{st}$
Epsilon	10^{-5}	16	10	$\mathbf{5}^{th}$
Zeta	10^{-5}	16	10	$\mathbf{4}^{th}$
OCR	10^{-5}	16	10	$\mathbf{2}^{nd}$
Face	10^{-5}	16	20	$\mathbf{4}^{th}$
DNA	10^{-3}	64	10	$\mathbf{2}^{nd}$
Webspam	10^{-5}	71,066	10	$\mathbf{4}^{th}$

Table 3.6: **Results of SGD-QN at the 1^{st} PASCAL Large Scale Learning Challenge.** Parameters and final ranks obtained in the "Wild Track". All competing algorithms were run by the organizers. (Note: the competition results were obtained with a preliminary version of SGD-QN. In particular the λ parameters listed above are different from the values used for all experiments in this paper and listed in Table 5.4.)

3.3 Summary

The SGD-QN algorithm strikes a good compromise for large scale application because it implements a quasi-Newton stochastic gradient descent while requiring low time and memory per iteration. As a result, SGD-QN empirically iterates nearly as fast as a first-order stochastic gradient descent but requires less iterations to achieve the same accuracy. SGD-QN won the "Wild Track" of the first PASCAL Large Scale Learning Challenge [Sonnenburg *et al.*, 2008].

In this chapter we also took care to precisely show how this performance is the result of a careful design taking into account the theoretical knowledge about second order SGD and a precise understanding of its algorithmic and implementation computational requirements.

4

Large-Scale SVMs for Binary Classification

Contents

Stochastic Gradient Descent provides efficient training methods for linear Support Vector Machines in a large-scale setup, as we showed in Chapter 3. However when it comes to non-linear kernels, SGD is no longer satisfactory because it can not exploit the sparsity of the kernel expansion (see equation 2.2) and suffers from the high complexity of the solution.

In this chapter we propose to study online learning methods for binary SVMs that work in the dual parameters space. We will demonstrate that this allows to deal efficiently with large-scale SVMs even when non-linear kernels are involved.

Given a training set $(x_1, y_1) \cdots (x_n, y_n)$, it has been shown in Section 2.1.1 that the dual of Support Vector Machines can take the form of the Quadratic Program:

$$\max_{\boldsymbol{\alpha}} \; D(\boldsymbol{\alpha}) = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j) \quad \text{with} \quad \begin{cases} \sum_i \alpha_i = 0 \\ A_i \leq \alpha_i \leq B_i \\ \quad A_i = \min(0, C y_i) \\ \quad B_i = \max(0, C y_i) \end{cases} \tag{4.1}$$

We also recall that we denote $\boldsymbol{g} = (g_1 \dots g_n)$ the gradient of the dual $D(\boldsymbol{\alpha})$ with

$$\forall k = 1, \dots n \; , \quad g_k \; = \; \frac{\partial D(\boldsymbol{\alpha})}{\partial \alpha_k} \; = \; y_k - \sum_i \alpha_i k(x_i, x_k) \; . \tag{4.2}$$

The first section of this chapter presents the Huller, a simple and efficient online kernel algorithm which eventually converges fast to the exact Hard Margin SVM classifier. Interestingly, it reaches competitive accuracies after a single pass over the training set. Unfortunately the Huller performs poorly on noisy data sets. In Section 4.2 is then introduced LaSVM. This online algorithm shares some desirable properties with the Huller: it reliably reaches competitive accuracies after performing a single pass over the training set and trains significantly faster than state-of-the-art SVM solvers. Besides, it solves the general Soft Margin SVM and thus handles noise properly. The online learning process of LaSVM raises some questions about the example selection. Section 4.3 addresses some of these by comparing several strategies for wisely choosing which training instance to process. We show that an active learning setup can decrease training duration and memory usage on large-scale problems, especially by increasing the sparsity of the kernel expansion. Finally Section 4.4 displays a novel duality lemma providing tracking guarantees for approximate incremental SVMs that compare with results about batch SVMs. This result also casts an interesting light on the online/active learning behavior of LaSVM.

The work presented in this chapter has been the object of two publications (e.g. [Bordes and Bottou, 2005] and [Bordes *et al.*, 2005]).

4.1 The Huller: an Efficient Online Kernel Algorithm

The Huller is a novel kernel classifier algorithm, whose basic optimization step is based on the geometrical formulation of SVMs. It works in online epochs over the training set, considering one example at a time. These properties cause the Huller to show an interesting behavior:

- Continued iterations of the algorithm converge to the exact Hard Margin SVM classifier.
- Like most SVM algorithms, and unlike most online kernel algorithms, it produces classifiers with a bias term. Removing the bias term is a known way to simplify the numerical aspects of SVMs (as for the methods discussed in Chapter 3). Unfortunately, this can also damage the classification accuracy [Keerthi *et al.*, 1999].
- Experiments on a relatively clean data set indicate that a single pass over the training set is sufficient to produce classifiers with competitive error rates, using a fraction of the time and memory required by state-of-the-art SVM solvers.

Section 4.1.1 reviews the geometric interpretation of SVMs. Section 4.1.2 presents a simple update rule for online algorithms that converge to the SVM solution and proposes a critical refinement. Section 4.1.3 reports experimental results. Finally Section 4.1.4 discusses the algorithm capabilities and limitations.

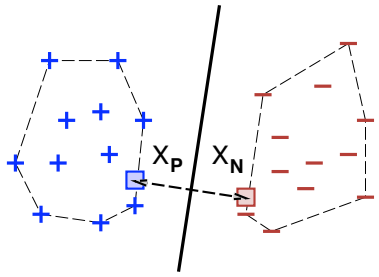


Figure 4.1: **Geometrical interpretation of Support Vector Machines.** The maximum margin hyperplane is the bisector of the segment linking $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$, the closest points belonging to the convex hulls formed by the examples of each class.

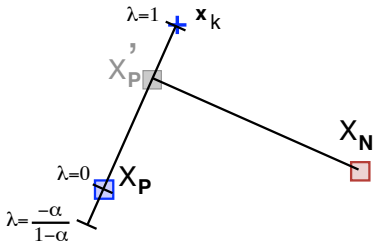


Figure 4.2: **Basic update of the Huller.** The new point $\boldsymbol{X'_P}$ is the point of segment $[\boldsymbol{X_P}, x_k]$ that minimizes the distance $\|\boldsymbol{X'_P} - \boldsymbol{X_N}\|^2$. It is defined using the λ parameter. A negative value for λ allows to remove vectors from the current solution.

4.1.1 Geometrical Formulation of SVMs

Figure 4.1 illustrates the geometrical formulation of SVMs [Bennett and Bredensteiner, 2000, Crisp and Burges, 2000]. Consider a training set composed of patterns x_i and corresponding classes $y_i = \pm 1$. When the training data is separable, the convex hulls formed by the positive and negative examples are disjoint. Consider two points $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$ belonging to each convex hull. Make them as close as possible without allowing them to leave their respective convex hulls. The median hyperplane of these two points is the maximum margin separating hyperplane.

The points $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$ can be parametrized as

$$\begin{array}{lll} \boldsymbol{X_P} = \sum_{i\in\mathcal{P}} \alpha_i x_i & \sum_{i\in\mathcal{P}} \alpha_i = 1 & \alpha_i \geq 0 \\ \boldsymbol{X_N} = \sum_{j\in\mathcal{N}} \alpha_j x_j & \sum_{j\in\mathcal{N}} \alpha_j = 1 & \alpha_j \geq 0 \end{array} \tag{4.3}$$

where sets $\mathcal{P}$ and $\mathcal{N}$ respectively contain the indices of the positive and negative examples. The optimal hyperplane is then obtained by solving

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{X_P} - \boldsymbol{X_N}\|^2 \tag{4.4}$$

under the constraints of the parametrization (4.3). The separating hyperplane is then represented by the following linear discriminant function:

$$f(x) = \langle (\boldsymbol{X_P} - \boldsymbol{X_N}), x \rangle + (\|\boldsymbol{X_N}\|^2 - \|\boldsymbol{X_P}\|^2)/2 \tag{4.5}$$

Since $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$ are represented as linear combinations of the training patterns, both the optimization criterion (4.4) and the discriminant function (4.5) can be expressed using dot products $\langle \cdot, \cdot \rangle$ between patterns. Arbitrary non linear classifiers can be derived by replacing these dot products by suitable kernel functions. For simplicity, we discuss the simple linear setup and leave the general kernel framework to the reader.

Equivalence to the Standard Formulation After a simple reorganization of the equality constraints, the optimization problem expressed by equations (4.3) and (4.4) can be summarized

as follows:

$$\max_{\boldsymbol{\alpha}} \left(-\frac{1}{2} \sum_{ij} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \right) \quad \text{with} \quad \left\{ \begin{array}{l} \forall i \;\; \alpha_i \geq 0 \\ \sum_i y_i \alpha_i = 0 \\ \sum_i \alpha_i = 2 \end{array} \right.$$

Observe that value 2 in the last constraint is arbitrary. We can replace this value by any positive constant K. This change simply rescales the coefficients $\boldsymbol{\alpha}$ without changing the position of the decision boundary. The Karush-Kuhn-Tucker theorem then states that $\boldsymbol{\alpha}$ are optimal if there is μ such that:

$$\forall i, \;\; \alpha_i \left(\mu - y_i \textstyle\sum_j y_j \alpha_j \langle x_i, x_j \rangle \right) = 0 \;, \text{ and } \sum \alpha_i = K \;, \quad \sum y_i \alpha_i = 0$$

Summing the first condition for all i yields: $K\mu = \sum_{ij} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle = \| \boldsymbol{X_P} - \boldsymbol{X_N} \|^2$.

This value is strictly positive *when the data is separable*. Then, for every positive constant K, there is a positive μ and vice-versa. Since we do not care about the value of K as long as it is positive, we can simply choose $\mu = 1$. The Karush-Kuhn-Tucker conditions then become:

$$\forall i, \;\; \alpha_i \left(1 - y_i \textstyle\sum_j y_j \alpha_j \langle x_i, x_j \rangle \right) = 0 \;, \quad \sum y_i \alpha_i = 0$$

We recognize the standard Hard Margin SVM [Vapnik, 1998] (similar to (2.10) with no upper bound on the values of the α_i):

$$\max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \frac{1}{2} \sum_{ij} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \quad \text{with} \quad \left\{ \begin{array}{l} \forall i \;\; \alpha_i \geq 0 \\ \sum_i y_i \alpha_i = 0 \end{array} \right.$$

The decision boundaries obtained by solving the problem expressed by equations (4.3) and (4.4) and by a Hard Margin SVM are thus identical.

4.1.2 The Huller Algorithm

Single Example Update

We now describe a first iterative algorithm that can be viewed as a simplification of the nearest point algorithms discussed in [Gilbert, 1966, Keerthi *et al.*, 1999]. The algorithm stores the position of points $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$ using the parametrization (4.3). Each iteration considers a training pattern x_k and updates the position of $\boldsymbol{X_P}$ (when $y_k = +1$) or $\boldsymbol{X_N}$ (when $y_k = -1$.)

Figure 4.2 illustrates the case where x_k is a positive example (negative examples are treated similarly). The new point $\boldsymbol{X'_P}$ is *a priori* the point of segment $[\boldsymbol{X_P}, x_k]$ that minimizes the distance $\| \boldsymbol{X'_P} - \boldsymbol{X_N} \|^2$. The new point $\boldsymbol{X'_P}$ can be expressed as $\boldsymbol{X'_P} = (1 - \lambda)\boldsymbol{X_P} + \lambda x_k$ with $0 \leq \lambda \leq 1$.

This first algorithm is flawed: suppose that the current $\boldsymbol{X_P}$ contains a non zero coefficient α_k that in fact should be zero. The algorithm cannot reduce this coefficient by selecting example x_k. It must instead select other positive examples and slowly erode the coefficient α_k by multiplying it by $(1 - \lambda)$. A simple fix was proposed by [Haffner, 2002]. If the coefficient α_k is strictly positive, we can safely let λ become slightly negative without leaving the convex hull. The revised constraints on λ are then $-\alpha_k/(1 - \alpha_k) \leq \lambda \leq 1$.

The optimal value of λ can be computed analytically by first computing the unconstrained optimum λ^{u}. When x_k is a positive example, solving $\left\langle (\boldsymbol{X_P} - \boldsymbol{X'_P}), (\boldsymbol{X_N} - \boldsymbol{X'_P}) \right\rangle = 0$, the orthogonality equation, for λ yields:

$$\lambda^{\mathrm{u}} = \frac{\langle (\boldsymbol{X_P} - \boldsymbol{X_N}), (\boldsymbol{X_P} - x_k) \rangle}{\| \boldsymbol{X_P} - x_k \|^2} = \frac{\| \boldsymbol{X_P} \|^2 - \langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle - \langle x_k, \boldsymbol{X_P} \rangle + \langle \boldsymbol{X_N}, x_k \rangle}{\| \boldsymbol{X_P} \|^2 + \| x_k \|^2 - 2 \langle x_k, \boldsymbol{X_P} \rangle} \tag{4.6}$$

Similarly, when x_k is a negative example, we obtain:

$$\lambda^{\mathrm{u}} = \frac{\langle (\boldsymbol{X_N} - \boldsymbol{X_P}), (\boldsymbol{X_N} - x_k) \rangle}{\|\boldsymbol{X_N} - x_k\|^2} = \frac{\|\boldsymbol{X_N}\|^2 - \langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle - \langle \boldsymbol{X_N}, x_k \rangle + \langle x_k, \boldsymbol{X_P} \rangle}{\|\boldsymbol{X_N}\|^2 + \|x_k\|^2 - 2 \langle \boldsymbol{X_N}, x_k \rangle} \tag{4.7}$$

A case by case analysis of the constraints shows that the optimal λ is:

$$\lambda = \min \left(1, \max \left(\frac{-\alpha_k}{1 - \alpha_k}, \lambda^{\mathrm{u}} \right) \right) \tag{4.8}$$

Both expressions (4.6) and (4.7) depend on the quantities $\|\boldsymbol{X_P}\|^2$, $\langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle$, and $\|\boldsymbol{X_N}\|^2$ whose computation could be expensive. Fortunately there is a simple way to avoid this calculation: in addition to points $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$, our algorithm also maintains three scalar variables containing the values of $\|\boldsymbol{X_P}\|^2$, $\langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle$, and $\|\boldsymbol{X_N}\|^2$. Their values are recursively updated after each iteration: when x_k is a positive example,

$$\begin{aligned} \|\boldsymbol{X'_P}\|^2 &= (1-\lambda)^2 \|\boldsymbol{X_P}\|^2 + 2\lambda(1-\lambda) \langle \boldsymbol{X_P}, x_k \rangle + \lambda^2 \|x_k\|^2 \\ \langle \boldsymbol{X_N}, \boldsymbol{X'_P} \rangle &= (1-\lambda) \langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle + \lambda \langle \boldsymbol{X_N}, x_k \rangle \\ \|\boldsymbol{X_N}\|^2 &= \|\boldsymbol{X_N}\|^2 \end{aligned} \tag{4.9}$$

and similarly, when x_k is a negative example,

$$\begin{aligned} \|\boldsymbol{X_P}\|^2 &= \|\boldsymbol{X_P}\|^2 \\ \langle \boldsymbol{X'_N}, \boldsymbol{X_P} \rangle &= (1-\lambda) \langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle + \lambda \langle x_k, \boldsymbol{X_P} \rangle \\ \|\boldsymbol{X'_N}\|^2 &= (1-\lambda)^2 \|\boldsymbol{X_N}\|^2 + 2\lambda(1-\lambda) \langle \boldsymbol{X_N}, x_k \rangle + \lambda^2 \|x_k\|^2 \end{aligned} \tag{4.10}$$

Algorithm 9 shows the resulting update algorithm. The cost of one update is dominated by the calculation of $\langle \boldsymbol{X_P}, x_k \rangle$ and $\langle \boldsymbol{X_N}, x_k \rangle$. This calculation requires the dot products between x_k and all the current support vectors, i.e. the training examples x_i with non zero coefficient α_i in the parametrization (4.3).

Algorithm 9 HullerUpdate(k)

1: Compute $\langle x_k, \boldsymbol{X_P} \rangle$, $\langle \boldsymbol{X_N}, x_k \rangle$, and $\|x_k\|^2$.
2: Compute λ^{u} using equations (4.6) or (4.7).
3: Compute λ using equation (4.8).
4: $\alpha_i \leftarrow (1 - \lambda)\alpha_i$ for all i such that $y_i = y_k$.
5: $\alpha_k \leftarrow \alpha_k + \lambda$.
6: Update $\|\boldsymbol{X_P}\|^2$, $\langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle$ and $\|\boldsymbol{X_N}\|^2$ using equation (4.9) or (4.10).

Algorithm 10 Huller

1: Initialize $\boldsymbol{X_P}$ and $\boldsymbol{X_N}$ by averaging a few points.
2: Compute initial $\|\boldsymbol{X_P}\|^2$, $\langle \boldsymbol{X_N}, \boldsymbol{X_P} \rangle$, and $\|\boldsymbol{X_N}\|^2$.
3: Pick a random p such that $\alpha_p = 0$.
4: HullerUpdate(p). ▷ Perform a Process operation
5: Pick a random r such that $\alpha_r \neq 0$.
6: HullerUpdate(r). ▷ Perform a Reprocess operation
7: Return to step 3.

Insertion and Removal

Simply repeating this update for random examples x_k works poorly. Most of the updates do nothing because they involve examples that are not support vectors and have no vocation to become support vectors. A closer analysis reveals that the update operation has two functions:

- Performing an update for an example x_k such that $\alpha_k = 0$ represents an attempt to insert this example into the current set of support vectors. This occurs when the optimal λ is greater than zero, that is, when the point x_k violates the SVM margin conditions. We term this kind of update a PROCESS.
- Performing an update for an example x_k such that $\alpha_k \neq 0$ will optimize the current solution and possibly remove this example from the current set of support vectors. The removal occurs when the optimal λ reaches its (negative) lower bound. We term this kind of update a REPROCESS.

Some work on kernel perceptrons [Crammer *et al.*, 2004] also rely on two separate processes to insert and remove support vectors from the expression of the current separating hyperplane. We discuss here a situation where both functions are implemented by the same update rule (depicted in Figure 4.2).

Picking the examples x_k randomly gives a disproportionate weight to the insertion function. The Huller algorithm (Algorithm 10) corrects this imbalance by allocating an equivalent computing time to both functions. First, it performs a PROCESS i.e. it picks a random example that is not a current support vector and attempts to insert it into the current set of support vectors. Second, it performs a REPROCESS i.e. it picks a random example that is a current support vector and attempts to remove it from the current set of support vectors. Implementing this simple PROCESS/REPROCESS principle has a dramatic effect on the convergence speed.

4.1.3 Experiments

The Huller algorithm was implemented in C and benchmarked against the state-of-the-art SVM solver LibSVM[1] on the well known MNIST[2] handwritten digit data set. All experiments were run with a RBF kernel with parameter $\gamma = 0.005$. Both LibSVM and the Huller implementations use the same code to compute the kernel values and similar strategies to cache the frequently used kernel values. The cache size was initially set to 256MB.

Figure 4.3 reports the experimental results on the ten problems consisting of classifying each of the ten digit category against all other categories. The Huller algorithm was run in epochs. Each epoch sequentially scans the randomly permuted MNIST training set and attempts to insert each example into the current set of support vectors (PROCESS operation in Algorithm 10). After each insertion attempt, the algorithm attempts to remove a random support vector (REPROCESS operation in Algorithm 10.) The Huller×1 results were obtained after a single epoch, that is after processing each example once. The Huller×2 results were obtained after two epochs. All results are averages over five runs.

The Huller×2 test errors (top left graph in Figure 4.3) closely match the LibSVM solution. This is confirmed by counting the number of support vectors (bottom left graph), The Huller×2 computing times usually are slightly shorter than the already fast LibSVM computing times (top right graph). The Huller×1 test errors (top left graph in Figure 4.3) are very close to both the Huller×2 and LibSVM test errors. Standard paired significance tests indicate that these small differences are not significant. This accuracy is achieved after less than half the LibSVM running

[1] `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
[2] `http://yann.lecun.com/exdb/mnist`

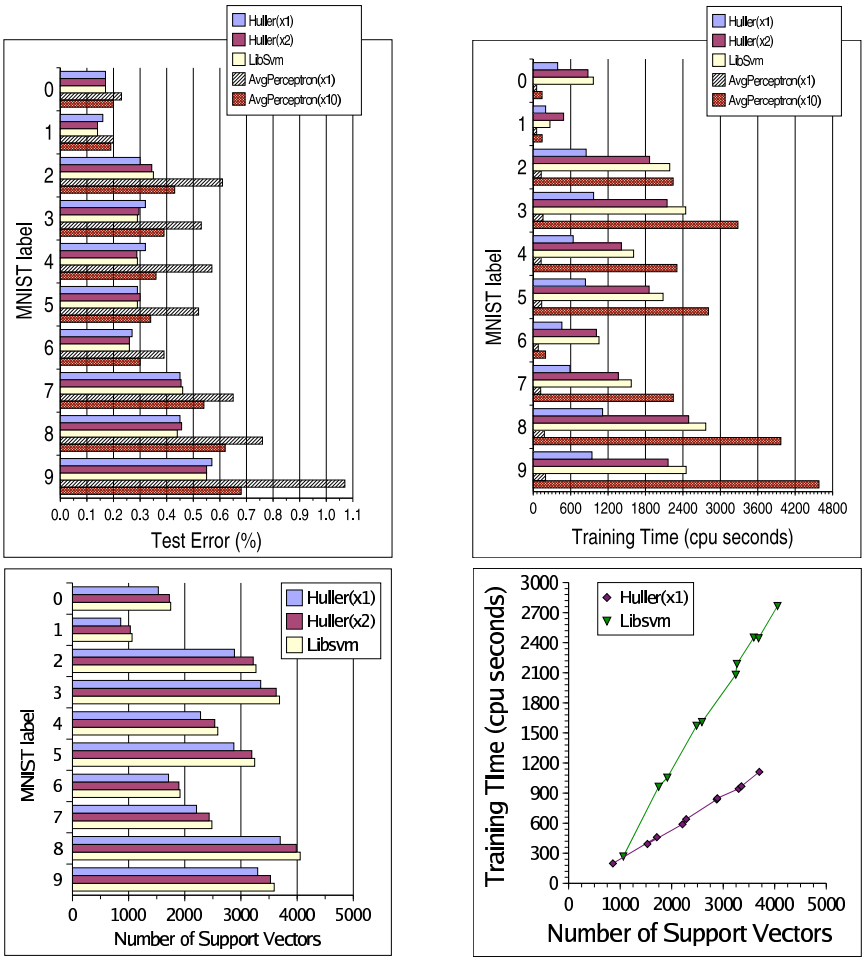


Figure 4.3: MNIST **results for the** Huller **(one and two epochs), for** LibSVM**, and for the** AvgPerc **(one and ten epochs).** Top left: test error accuracies. Top right: training time. Bottom left: number of support vectors. Bottom right: training time as a function of the number of support vectors: LibSVM and the Huller have a linear behavior but the latter is more efficient.

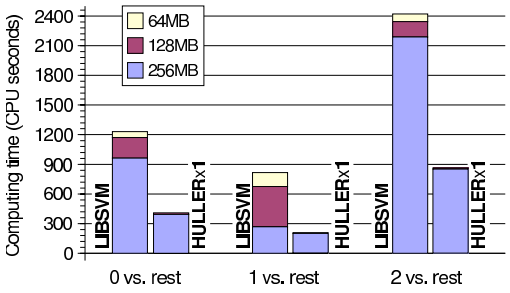


Figure 4.4: **Computing times with various cache sizes.** Each color indicates the additional time required when reducing the cache size. The Huller times remain virtually unchanged.

time, and, more importantly, after a single sequential pass over the training examples. The Huller×1 always yields a slightly smaller number of support vectors (bottom left graph). We believe that a single Huller epoch fails to insert a few examples that appear as support vectors in the SVM solution. A second epoch recaptures most missing examples.

Neither the Huller×1 or Huller×2 experiments yield the exact SVM solution. On this data set, the Huller typically reaches the SVM solution after five epochs. The corresponding computing times are not competitive with those achieved by LibSVM.

These results should also be compared with results obtained with a theoretically justified kernel perceptron algorithm. Figure 4.3 contains results obtained with the AvgPerc [Freund and Schapire, 1998] using the same kernel and cache size. The first epoch runs very quickly but does not produce competitive error rates. The AvgPerc approaches[3] the LibSVM or Huller×1 accuracies after ten epochs[4]. The corresponding training times stress the importance of the kernel cache size. When the cache can accommodate the dot products of all examples with all support vectors, additional epochs require very little computation. When this is not the case, the AvgPerc times are not competitive.

Figure 4.4 shows how reducing the cache size affects the computing time. Whereas LibSVM experiences significantly increased training times, the Huller training times are essentially unchanged. The most dramatic case is the separation of digit "1" versus all other categories. The initial 256MB cache size is sufficient for holding all the kernel values required by LibSVM. Under these condition, LibSVM runs almost as quickly as the Huller×1. Reducing the kernel cache size to 128MB doubles the LibSVM training time and does not change the Huller training times.

A detailed analysis of the algorithms indicates that LibSVM runs best when the cache contains all the dot products involving a potential support vector and an arbitrary example: memory requirements grow with both the number of support vectors and the number of training examples. The Huller runs best when the cache contains all the dot products involving two potential support vectors: the memory requirements grow with the number of support vectors only. This indicates that the Huller is best suited for problems involving a large separable training set.

4.1.4 Discussion

The Huller processes many more examples during the very first training stages. After processing the first pair of examples, the SMO core of LibSVM must compute 120000 dot products to update the example gradients and choose the next pair. During the same time, the Huller processes at least 500 examples. By the time LibSVM has reached the fifth pair of examples, the Huller has processed a minimum of 1500 fresh examples. Online kernel classifiers without removal step tend to slow down sharply because the number of support vectors increases quickly. The removal step ensures that the number of current support vectors does not significantly exceed the final number of support vectors.

This does not mean that LibSVM computes useless dot products. To simply assert that the SVM solution has been reached, any SVM solver needs the values of every dot product appearing in the SVM Karush-Kuhn-Tucker conditions. Depending on the problem, modern SMO solvers request no more than 10% to 40% additional dot products.

To attain the exact SVM solution with confidence, the Huller also must compute all the dot products it did not compute in the early stages. On the other hand, when the kernel cache size is large enough, LibSVM already knows these values and can use this rich local information to move more judiciously. This is why LibSVM outperforms the huller in the final stages of the

[3]This is consistent with the empirical results reported in [Freund and Schapire, 1998] (Table 3).

[4]The Averaged Perceptron theoretical guarantees only hold for a single epoch.

optimization. Nevertheless, the Huller produces competitive classifiers well before reaching the point where it gets outpaced by state-of-the-art SVM optimization packages such as LibSVM.

4.2 Online LaSVM

The Huller addresses the Hard-Margin SVM problem and therefore performs poorly on noisy data sets [Cortes and Vapnik, 1995]. Even if many online kernel classifiers share this limitation, this remains penalizing on most tasks. This section proposes a novel algorithm named LaSVM that furthers ideas presented in the previous section but also fixes the limitations of the Huller.

Following the principle used for the Huller, LaSVM is an online kernel classifier which alternates two kinds of direction searches named Process and Reprocess. Each direction search involves a pair of examples. Direction searches of the Process kind involve at least one example that is not a support vector and can potentially change its coefficient to make it a support vector. Direction searches of the Reprocess kind involve two examples that are already support vectors and can potentially zero their coefficients to remove them from the kernel expansion. Besides, LaSVM is also a reorganization of the SMO sequential direction searches and, as such, converges to the solution of the SVM QP problem (4.1). Section 4.2.1 details the LaSVM operations.

Compared to basic kernel perceptrons [Aizerman *et al.*, 1964, Freund and Schapire, 1998], the LaSVM algorithm features a removal step and gracefully handles noisy data. Compared to kernel perceptrons with removal steps [Crammer *et al.*, 2004, Weston *et al.*, 2005], LaSVM converges to the known SVM solution. Compared to a traditional SVM solver [Platt, 1999, Chang and Lin, 2001 2004, Collobert and Bengio, 2001], LaSVM brings the computational benefits and the flexibility of online learning algorithms. In addition, experimental evidence on multiple data sets (Section 4.2.5) indicates that LaSVM reliably reaches competitive test error rates after performing a single pass over the training set. It uses less memory and trains significantly faster than state-of-the-art SVM solvers.

4.2.1 Building Blocks

The LaSVM algorithm maintains three essential pieces of information: the set $\mathcal{S}$ of potential support vector indices, the coefficients α_i of the current kernel expansion, and the partial derivatives g_i defined in (4.2). Variables α_i and g_i contain meaningful values when $i \in \mathcal{S}$ only. The coefficient α_i are assumed to be null if $i \notin \mathcal{S}$. On the other hand, set $\mathcal{S}$ might contain a few indices i such that $\alpha_i = 0$.

The two basic operations of the online LaSVM algorithm correspond to steps 2 and 3 of the SMO algorithm (see Algorithm 1 in Section 2.1.2). These two operations differ from each other because they have different ways to select τ-violating pairs.

The first operation, Process (Algorithm 11), attempts to insert example $k \notin \mathcal{S}$ into the set of current support vectors. In the online setting this is used to process a new example at time t. It first adds example $k \notin \mathcal{S}$ into $\mathcal{S}$ (step 1-2). Then it searches a second example in $\mathcal{S}$ to find the τ-violating pair with maximal gradient (steps 3-4) and performs a direction search (step 5).

The second operation, Reprocess (Algorithm 12), removes some elements from $\mathcal{S}$. It first searches the τ-violating pair of elements of $\mathcal{S}$ with maximal gradient (steps 1-2), and performs a direction search (step 3). Then it removes blatant non support vectors (step 4). Finally it computes two useful quantities: the bias term b of the decision function (2.2) and the gradient δ of the most τ-violating pair in $\mathcal{S}$.

Algorithm 11 PROCESS(k)

Bail out if $k \in \mathcal{S}$.
$\alpha_k \leftarrow 0$, $g_k \leftarrow y_k - \sum_{s \in \mathcal{S}} \alpha_s \, k(x_k, x_s)$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$
if $y_k = +1$ **then**
  $i \leftarrow k$, $j \leftarrow \arg\min_{s \in S} g_s$ with $\alpha_s > A_s$
else
  $j \leftarrow k$, $i \leftarrow \arg\max_{s \in S} g_s$ with $\alpha_s < B_s$
end if
Bail out if (i, j) is not a τ-violating pair.
$\lambda \leftarrow \min \left\{ \dfrac{g_i - g_j}{k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)}, \; B_i - \alpha_i, \; \alpha_j - A_j \right\}$
$\alpha_i \leftarrow \alpha_i + \lambda$, $\alpha_j \leftarrow \alpha_j - \lambda$
$g_s \leftarrow g_s - \lambda \left(k(x_i, x_s) - k(x_j, x_s) \right) \quad \forall \, s \in \mathcal{S}$

Algorithm 12 REPROCESS

$i \leftarrow \arg\max_{s \in \mathcal{S}} g_s$ with $\alpha_s < B_s$
$j \leftarrow \arg\min_{s \in \mathcal{S}} g_s$ with $\alpha_s > A_s$
Bail out if (i, j) is not a τ-violating pair.
$\lambda \leftarrow \min \left\{ \dfrac{g_i - g_j}{k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)}, \; B_i - \alpha_i, \; \alpha_j - A_j \right\}$
$\alpha_i \leftarrow \alpha_i + \lambda$, $\alpha_j \leftarrow \alpha_j - \lambda$
$g_s \leftarrow g_s - \lambda \left(k(x_i, x_s) - k(x_j, x_s) \right) \quad \forall \, s \in \mathcal{S}$
$i \leftarrow \arg\max_{s \in \mathcal{S}} g_s$ with $\alpha_s < B_s$
$j \leftarrow \arg\min_{s \in \mathcal{S}} g_s$ with $\alpha_s > A_s$
for all $s \in \mathcal{S}$ such that $\alpha_s = 0$ **do**
  if $y_s = -1$ and $g_s \geq g_i$ **then**
    $\mathcal{S} = \mathcal{S} - \{s\}$
  else if $y_s = +1$ and $g_s \leq g_j$ **then**
    $\mathcal{S} = \mathcal{S} - \{s\}$
  end if
end for
$b \leftarrow (g_i + g_j)/2$, $\delta \leftarrow g_i - g_j$

4.2.2 Scheduling

After initializing the state variables (step 1), the online LaSVM algorithm alternates PROCESS and REPROCESS a predefined number of times (step 2). Then it simplifies the kernel expansion by running REPROCESS to remove all τ-violating pairs remaining in the kernel expansion (step 3). It is presented in Algorithm 13.

LaSVM can be used in the online setup where one is given a continuous stream of fresh random examples. The online iterations process fresh training examples as they come. LaSVM can also be used as a stochastic optimization algorithm in the batch setup where the complete training set is available before hand. Each iteration randomly picks an example from the training set.

In practice we run the LaSVM online iterations in epochs. Each epoch sequentially visits all the randomly shuffled training examples. After a predefined number P of epochs, we perform the (optional) finishing step. A single epoch is consistent with the use of LaSVM in the online setup.

Algorithm 13 LaSVM

1: **Initialization**:
Seed $\mathcal{S}$ with a few examples of each class.
Set $\boldsymbol{\alpha} \leftarrow \mathbf{0}$ and compute the initial gradient $\boldsymbol{g}$ (equation 4.2)
2: **Online Iterations**:
3: Repeat a predefined number of times:
- Pick an example k_t
- Run Process(k_t).
- Run Reprocess once.

4: **Finishing**:
Repeat Reprocess until $\delta \leq \tau$.

Multiple epochs are consistent with the use of LaSVM as a stochastic optimization algorithm in the batch setup.

4.2.3 Convergence and Complexity

Let us first ignore the finishing step (step 3) and assume that online iterations (step 2) are repeated indefinitely. Suppose that there are remaining τ-violating pairs at iteration T.

a.) If there are τ-violating pairs (i, j) such that $i \in \mathcal{S}$ and $j \in \mathcal{S}$, one of them will be exploited by the next Reprocess.

b.) Otherwise, if there are τ-violating pairs (i, j) such that $i \in \mathcal{S}$ or $j \in \mathcal{S}$, each subsequent Process has a chance to exploit one of them. The intervening Reprocess do nothing because they bail out at step 2.

c.) Otherwise, all τ-violating pairs involve indices outside $\mathcal{S}$. Subsequent calls to Process and Reprocess bail out until we reach a time $t > T$ such that $k_t = i$ and $k_{t+1} = j$ for some τ-violating pair (i, j). The first Process then inserts i into $\mathcal{S}$ and bails out. The following Reprocess bails out immediately. Finally the second Process locates pair (i, j).

This case is not important in practice. There usually is a support vector $s \in \mathcal{S}$ such that $A_s < \alpha_s < B_s$. We can then write $g_i - g_j = (g_i - g_s) + (g_s - g_j) \leq 2\tau$ and conclude that we already have reached a 2τ-approximate solution.

The LaSVM online iterations therefore work like the SMO algorithm. Remaining τ-violating pairs are sooner or later exploited by either Process or Reprocess. As soon as a τ-approximate solution is reached, the algorithm stops updating the coefficients $\boldsymbol{\alpha}$. Theorem 28 in the Appendix B gives more precise convergence results for this stochastic algorithm.

The finishing step (step 3) is only useful when one limits the number of online iterations. Running LaSVM usually consists in performing a predefined number P of epochs and running the finishing step. Each epoch performs n online iterations by sequentially visiting the randomly shuffled training examples. Empirical evidence suggests indeed that a *single epoch* yields a classifier almost as good as the SVM solution.

Computational Cost of LaSVM Both Process and Reprocess require a number of operations proportional to the number S of support vectors in set $\mathcal{S}$. Performing P epochs of online iterations requires a number of operations proportional to $n\,P\,\bar{S}$. The average number $\bar{S}$ of support vectors scales no more than linearly with n because each online iteration brings at most one new support

vector. The asymptotic cost therefore grows like n^2 at most. The finishing step is similar to running a SMO solver on a SVM problem with only S training examples. We recover here the n^2 to n^3 behavior of standard SVM solvers (as discussed in Section 2.1.1).

Online algorithms access kernel values with a very specific pattern. Most of the kernel values accessed by Process and Reprocess involve only support vectors from set $\mathcal{S}$. Only Process on a new example x_{k_t} accesses S fresh kernel values $K(x_{k_t}, x_i)$ for $i \in \mathcal{S}$.

4.2.4 Implementation Details

Our LaSVM implementation reorders the examples after every Process or Reprocess to ensure that the current support vectors come first in the reordered list of indices. The kernel cache records truncated rows of the reordered kernel matrix. SVMLight [Joachims, 1999] and LibSVM [Chang and Lin, 2001 2004] also perform such reordering, but do so rather infrequently. The reordering overhead is acceptable during the online iterations because the computation of fresh kernel values takes much more time.

Reordering examples during the finishing step was more problematic. We eventually deployed an adaptation of the *shrinking* heuristic [Joachims, 1999] for the finishing step only. The set $\mathcal{S}$ of support vectors is split into an active set S_a and an inactive set S_i. All support vectors are initially active. The Reprocess iterations are restricted to the active set S_a and do not perform any reordering. About every 1000 iterations, support vectors that hit the boundaries of the box constraints are either removed from the set $\mathcal{S}$ of support vectors or moved from the active set $\mathcal{S}_a$ to the inactive set $\mathcal{S}_i$. When all τ-violating pairs of the active set are exhausted, the inactive set examples are transferred back into the active set. The process continues as long as the merged set contains τ-violating pairs.

A C implementation of LaSVM, featuring the kernel cache, is freely available on the `mloss.org` website under the `GNU Public License` (go to `http://mloss.org/software/view/23/`).

4.2.5 Experiments

MNIST Experiments

The online LaSVM was first evaluated on the MNIST[5] handwritten digit data set, that we already used for benchmarking the Huller. Computing kernel values for this data set is relatively expensive because it involves dot products of 784 gray level pixel values. In the experiments reported below, all algorithms use the same code for computing kernel values. The ten binary classification tasks consist of separating each digit class from the nine remaining classes. All experiments use RBF kernels with $\gamma = 0.005$ and the same training parameters $C = 1000$ and $\tau = 0.001$. Unless indicated otherwise, the kernel cache size is 256MB.

LaSVM vs Sequential Minimal Optimization Baseline results were obtained by running the state-of-the-art SMO solver LibSVM [Chang and Lin, 2001 2004]. The resulting classifier accurately represents the SVM solution.

Two sets of results are reported for LaSVM. The LaSVM×1 results were obtained by performing a single epoch of online iterations: each training example was processed exactly once during a single sequential sweep over the training set. The LaSVM×2 results were obtained by performing two epochs of online iterations.

Figures 4.5 and 4.6 show the resulting test errors and training times. LaSVM×1 runs about three times faster than LibSVM and yields test error rates very close to the LibSVM results. Stan-

[5] `http://yann.lecun.com/exdb/mnist`

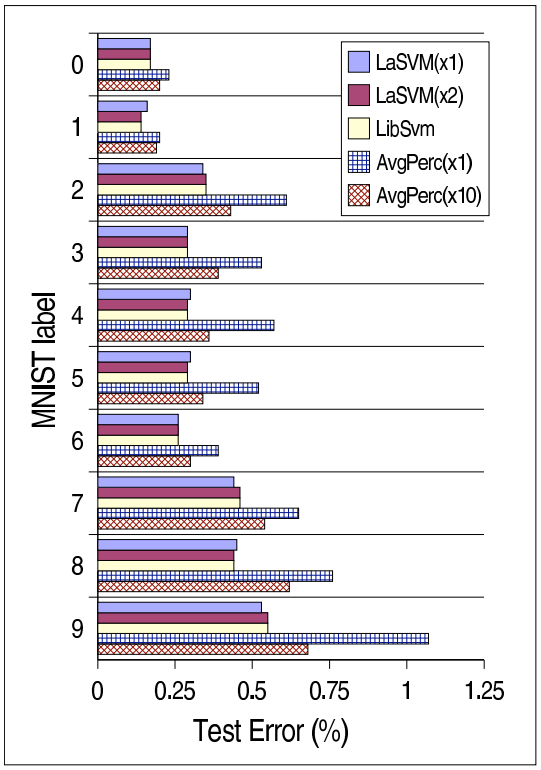


Figure 4.5: **Compared test error rates for the ten MNIST binary classifiers.**

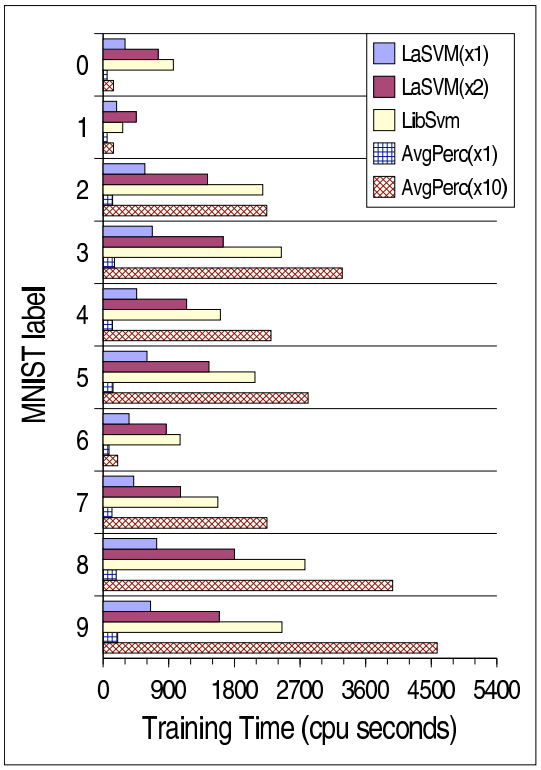


Figure 4.6: **Compared training times for the ten MNIST binary classifiers.**

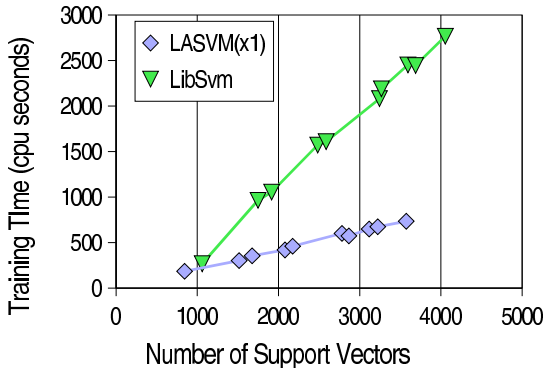


Figure 4.7: **Training time as a function of the number of support vectors.**

Algorithm	Error	Time
LibSVM	**1.36%**	17400s
LaSVM×1	1.42%	**4950s**
LaSVM×2	**1.36%**	12210s

Table 4.1: **Multiclass errors and training times for the MNIST data set.**

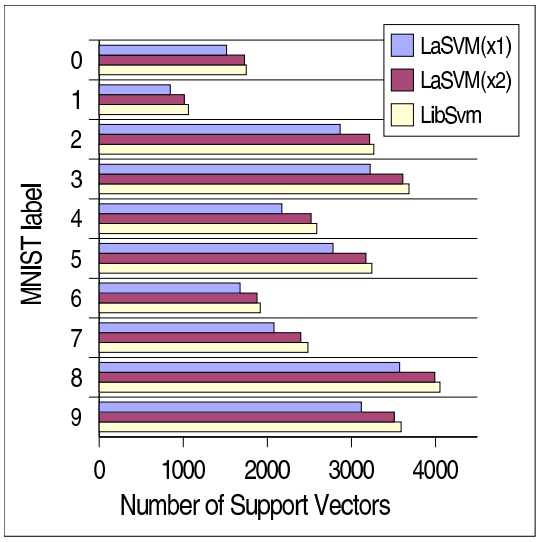


Figure 4.8: **Compared numbers of support vectors for the ten** MNIST **binary classifiers.**

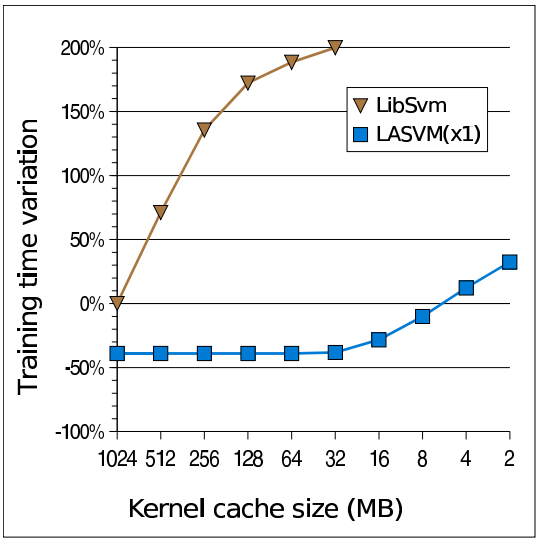


Figure 4.9: **Training time variation as a function of the cache size.** Relative changes with respect to the 1GB LibSVM times are averaged over all ten MNIST classifiers.

dard paired significance tests indicate that these small differences are not significant. LaSVM×2 usually runs faster than LibSVM and very closely tracks the LibSVM test errors.

Neither the LaSVM×1 or LaSVM×2 experiments yield the exact SVM solution. On this data set, LaSVM reaches the exact SVM solution after about five epochs. The first two epochs represent the bulk of the computing time. The remaining epochs run faster when the kernel cache is large enough to hold all the dot products involving support vectors. Yet the overall optimization times are not competitive with those achieved by LibSVM.

Figure 4.7 shows the training time as a function of the final number of support vectors for the ten binary classification problems. Both LibSVM and LaSVM×1 show a linear dependency. The online LaSVM algorithm seems more efficient overall.

Table 4.1 shows the multiclass error rates and training times obtained by combining the ten classifiers using the well known 1-versus-rest scheme [Schölkopf and Smola, 2002]. LaSVM×1 provides almost the same accuracy with much shorter training times. LaSVM×2 reproduces the LibSVM accuracy with slightly shorter training time.

Figure 4.8 shows the resulting number of support vectors. A single epoch of the online LaSVM algorithm gathers most of the support vectors of the SVM solution computed by LibSVM. The first iterations of the online LaSVM might indeed ignore examples that later become support vectors. Performing a second epoch captures most of the missing support vectors.

LaSVM vs the Averaged Perceptron The computational advantage of LaSVM relies on its apparent ability to match the SVM accuracies after a single epoch. Therefore it must be compared with algorithms such as the Averaged Perceptron [Freund and Schapire, 1998] that provably match well known upper bounds on the SVM accuracies. The AvgPerc×1 results in Figures 4.5 and 4.6 were obtained after running a single epoch of the Averaged Perceptron.

Although the computing times are very good, the corresponding test errors are not competitive with those achieved by either LibSVM or LaSVM. [Freund and Schapire, 1998] suggest that the Averaged Perceptron approaches the actual SVM accuracies after 10 to 30 epochs. Doing so no longer provides the theoretical guarantees. The AvgPerc×10 results in Figures 4.5 and 4.6 were obtained after ten epochs. Test error rates indeed approach the SVM results. The corresponding training times are no longer competitive.

LaSVM vs the Huller Both LaSVM and the Huller have been evaluated on MNIST with the same kernel (and on similar computers). We can then perform a fair comparison of results displayed in Figure 4.3 for the Huller and in Figures 4.5, 4.6, 4.7 and 4.8 for LaSVM. A first glance shows that both algorithms perform in like manner: same good scaling behavior in one pass, cheap memory usage and comparable accuracies. LaSVM is maybe slightly faster.

The main difference between them is that LaSVM trains Soft Margin SVMs and this is crucial to deal with noisy data. On MNIST, we set C to a high value (1000) because this data set is not very noisy. As a result, being restricted to the Hard Margin formulation is not really damaging for the Huller on it. However, in the following, we display experimental results of LaSVM on greatly noisier benchmarks (requiring much lower C values, see Table 4.2). Reach competitive error rates on them would be impossible for the Huller.

Impact of the Kernel Cache Size Training times stress the importance of the kernel cache size. Figure 4.6 shows how the AvgPerc×10 runs much faster on problems 0, 1, and 6. This is happening because the cache is large enough to accommodate the dot products of all examples with all support vectors. Each repeated iteration of the Average Perceptron requires very few additional kernel evaluations. This is much less likely to happen when the training set size increases. Computing times then increase drastically because repeated kernel evaluations become necessary.

Figure 4.9 compares how the LibSVM and LaSVM×1 training times change with the kernel cache size. The vertical axis reports the relative changes with respect to LibSVM with one gigabyte of kernel cache. These changes are averaged over the ten MNIST classifiers. The plot shows how LaSVM tolerates much smaller caches. On this problem, LaSVM *with a 8MB cache runs slightly faster than* LibSVM *with a 1024MB cache.*

Useful orders of magnitude can be obtained by evaluating how large the kernel cache must be to avoid the systematic recomputation of dot-products. Following the notations of Section 2.1.1, let n be the number of examples, S be the number of support vectors, and $R \leq S$ the number of support vectors such that $0 < |\alpha_i| < C$.

- In the case of LibSVM, the cache must accommodate about $n\,R$ terms. Indeed, each SMO iteration selects one example among the R free support vectors and performs n distinct dot-products with this selected example. As all SMO iterations are conducted several times during training, the cache needs to store $n\,R$ kernel values to be optimal.
- To perform a *single* LaSVM epoch, the cache must only accommodate about $S\,R$ terms. Since the examples are visited only once, the dot-products computed by a Process operation can only be reused by subsequent Reprocess operations. The cache must then concentrate on them. As (1) the examples selected by Reprocess are usually chosen among the R free support vectors, and (2) for each selected example, Reprocess needs one distinct dot-product per support vector in set $\mathcal{S}$, the cache needs to store $S\,R$ kernel values.
- To perform *multiple* LaSVM epochs, the cache must accommodate about $n\,S$ terms: the dot-products computed by processing a particular example are reused when processing

the same example again in subsequent epochs. This also applies to multiple Averaged Perceptron epochs.

An efficient single epoch learning algorithm is therefore very desirable when one expects S to be much smaller than n. Unfortunately, this may not be the case when the data set is noisy. The next section presents results obtained in such less favorable conditions.

Multiple Data Sets Experiments

Further experiments were carried out with a collection of standard data sets representing diverse noise conditions, training set sizes, and input dimensionality. Figure 4.2 presents these data sets and the parameters used for the experiments. Kernel computation times for these data sets are extremely fast. The data either has low dimensionality or can be represented with sparse vectors. For instance, computing kernel values for two REUTERS documents only involves words common to both documents (excluding stop words). The FOREST experiments use a kernel implemented with hand optimized assembly code [Graf *et al.*, 2005].

Table 4.3 compares the solutions returned by LaSVM×1 and LibSVM. The LaSVM×1 experiments call the kernel function much less often, but do not always run faster. The fast kernel computation times expose the relative weakness of our kernel cache implementation. The LaSVM×1 accuracies are very close to the LibSVM accuracies. The number of support vectors is always slightly smaller.

LaSVM×1 essentially achieves consistent results over very diverse data sets, after performing one single epoch over the training set only. In this situation, the LaSVM PROCESS function gets only one chance to take a particular example into the kernel expansion and potentially make it a support vector. The conservative strategy would be to take all examples and sort them out during the finishing step. The resulting training times would always be worse than LibSVM's because the finishing step is itself a simplified SMO solver. Therefore LaSVM online iterations are able to very quickly discard a large number of examples with a high confidence. This process is not perfect because we can see that the LaSVM×1 number of support vectors are smaller than LibSVM's. Some good support vectors are discarded erroneously.

Figure 4.4 reports the relative variations of the test error, number of support vectors, and training time measured before and after the finishing step. The online iterations pretty much select the right support vectors on clean data sets such as WAVEFORM, REUTERS or USPS, and the finishing step does very little. On the other problems the online iterations keep much more examples as potential support vectors. The finishing step significantly improves the accuracy on noisy data sets such as BANANA, ADULT or USPS+N, and drastically increases the computation time on data sets with complicated decision boundaries such as BANANA or FOREST.

The Collection of Potential Support Vectors The final step of the REPROCESS operation computes the current value of the kernel expansion bias b and the stopping criterion δ.

$$\begin{aligned} g_{\max} &= \max_{s\in\mathcal{S}} g_s \quad \text{with } \alpha_s < B_s & \qquad b &= \frac{g_{\max}+g_{\min}}{2} \\ g_{\min} &= \min_{s\in\mathcal{S}} g_s \quad \text{with } \alpha_s > A_s & \qquad \delta &= g_{\max}-g_{\min} \end{aligned} \tag{4.11}$$

The quantities $g_{\min}$ and $g_{\max}$ can be interpreted as bounds for the decision threshold b. The quantity δ then represents an uncertainty on the decision threshold b.

The quantity δ also controls how LaSVM collects potential support vectors. The definition of PROCESS and the equality (4.2) indicate indeed that PROCESS(k) adds the support vector x_k to

	Train Size	Test Size	γ	C	Cache	τ	Notes
Waveform[1]	4000	1000	0.05	1	40M	0.001	Artificial data, 21 dims.
Banana[1]	4000	1300	0.5	316	40M	0.001	Artificial data, 2 dims.
Reuters[2]	7700	3299	1	1	40M	0.001	Topic "moneyfx" vs. rest.
USPS[3]	7329	2000	2	1000	40M	0.001	Class "0" vs. rest.
USPS+N[3]	7329	2000	2	10	40M	0.001	10% training label noise.
Adult[3]	32562	16282	0.005	100	40M	0.001	As in [Platt, 1999].
Forest[3] (100k)	100000	50000	1	3	512M	0.001	As in [Collobert *et al.*, 2002].
Forest[3] (521k)	521012	50000	1	3	1250M	0.01	As in [Collobert *et al.*, 2002].

[1] `http://mlg.anu.edu.au/~raetsch/data/index.html`
[2] `http://www.daviddlewis.com/resources/testcollections/reuters21578`
[3] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases`

Table 4.2: **Data sets discussed in Section 4.2.5.**

	LibSVM				LaSVM×1			
	Error	SV	KCalc	Time	Error	SV	KCalc	Time
Waveform	8.82%	1006	4.2M	3.2s	**8.68%**	**948**	**2.2M**	**2.7s**
Banana	9.96%	873	6.8M	9.9s	9.98%	869	6.7M	10.0s
Reuters	2.76%	1493	11.8M	**24s**	2.76%	1504	**9.2M**	31.4s
USPS	0.41%	236	1.97M	**13.5s**	0.43%	**201**	**1.08M**	15.9s
USPS+N	**0.41%**	2750	63M	305s	0.53%	**2572**	**20M**	**178s**
Adult	14.90%	11327	1760M	1079s	14.94%	11268	**626M**	**809s**
Forest (100k)	**8.03%**	43251	27569M	14598s	8.15%	**41750**	**18939M**	**10310s**
Forest (521k)	4.84%	124782	316750M	159443s	4.83%	122064	**188744M**	**137183s**

Table 4.3: **Comparison of LibSVM versus LaSVM×1** Test error rates (Error), number of support vectors (SV), number of kernel calls (KCalc), and training time (Time). Bold characters indicate significative differences.

	Relative Variation		
	Error	SV	Time
Waveform	-0%	-0%	+4%
Banana	-79%	-74%	+185%
Reuters	0%	-0%	+3%
USPS	0%	-2%	+0%
USPS+N	-67%	-33%	+7%
Adult	-13%	-19%	+80%
Forest (100k)	-1%	-24%	+248%
Forest (521k)	-2%	-24%	+84%

Table 4.4: **Influence of the finishing step** on test error, number of support vectors and training time. This can be highly beneficial (USPS+N) or a waste of time (Forest (100k)).

the kernel expansion if and only if:

$$y_k\, f(x_k) < 1 + \frac{\delta}{2} - \tau \tag{4.12}$$

When $\boldsymbol{\alpha}$ is optimal, the uncertainty δ is zero, and this condition matches the Karush-Kuhn-Tucker condition for support vectors $y_k\, f(x_k) \leq 1$.

Intuitively, relation (4.12) describes how Process collects potential support vectors that are compatible with the current uncertainty level δ on the threshold b. Simultaneously, the Reprocess operations reduce δ and discard the support vectors that are no longer compatible with this reduced uncertainty.

The online iterations of the LaSVM algorithm make equal numbers of Process and Reprocess for purely heuristic reasons. Nothing guarantees that this is the optimal proportion. The results reported in Figure 4.10 clearly suggest to investigate this arbitrage more closely.

Variations on Reprocess Experiments were carried out with a slightly modified LaSVM algorithm: instead of performing a single Reprocess, the modified online iterations repeatedly run Reprocess until the uncertainty δ becomes smaller than a predefined threshold $\delta_{\max}$.

Figure 4.10 reports comparative results for the Banana data set. Similar results were obtained with other data sets. The three plots report test error rates, training time, and number of support vectors as a function of $\delta_{\max}$. These measurements were performed after one epoch of online iterations without finishing step, and after one and two epochs followed by the finishing step. The corresponding LibSVM figures are indicated by large triangles on the right side.

Regardless of $\delta_{\max}$, the SVM test error rate can be replicated by performing two epochs followed by a finishing step. However, this does not guarantee that the optimal SVM solution has been reached. Large values of $\delta_{\max}$ essentially correspond to the unmodified LaSVM algorithm. Small values of $\delta_{\max}$ considerably increases the computation time because each online iteration calls Reprocess many times in order to sufficiently reduce δ. Small values of $\delta_{\max}$ also remove the LaSVM ability to produce a competitive result after a single epoch followed by a finishing step. The additional optimization effort discards support vectors more aggressively. Additional epochs are necessary to recapture the support vectors that should have been kept.

There clearly is a sweet spot around $\delta_{\max} = 3$ when one epoch of online iterations alone almost match the SVM performance and also makes the finishing step very fast. This sweet spot is difficult to find in general. If $\delta_{\max}$ is a little bit too small, we must make one extra epoch. If $\delta_{\max}$ is a little bit too large, the algorithm behaves like the unmodified LaSVM. Short of a deeper understanding of these effects, the unmodified LaSVM seems to be a robust compromise.

SimpleSVM The right side of each plot in Figure 4.10 corresponds to an algorithm that optimizes the coefficient of the current support vectors at each iteration. This is closely related to the SimpleSVM algorithm [Vishwanathan *et al.*, 2003]. Both LaSVM and SimpleSVM update a current kernel expansion by adding or removing one or two support vectors at each iteration. The two key differences are the numerical objective of these updates and their costs.

Whereas each SimpleSVM iteration seeks the optimal solution of the SVM QP problem restricted to the current set of support vectors, the LaSVM online iterations merely attempt to improve the value of the dual objective function $D(\boldsymbol{\alpha})$. As a a consequence, LaSVM needs a finishing step and the SimpleSVM does not. On the other hand, Figure 4.10 suggests that seeking the optimum at each iteration discards support vectors too aggressively to reach competitive accuracies after a single epoch. Moreover, we propose in Section 4.4 an analysis showing that, without explicitly seeking the true optimum at each step, LaSVM fulfills an approximate optimality criterion on the course of learning.

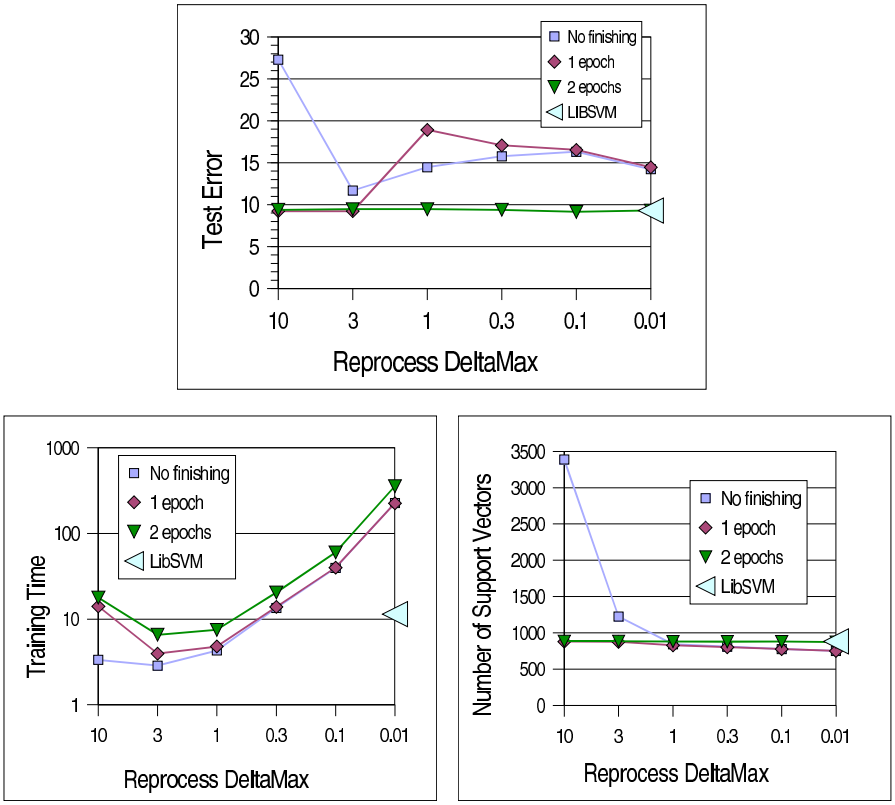


Figure 4.10: **Impact of additional** REPROCESS **measured on** BANANA **data set.** During the LaSVM online iterations, calls to REPROCESS are repeated until $\delta < \delta_{\max}$.

Each SimpleSVM iteration updates the current kernel expansion using rank 1 matrix updates [Cauwenberghs and Poggio, 2001] whose computational cost grows as the square of the number of support vectors. LaSVM performs these updates using SMO direction searches whose cost grows linearly with the number of examples. Rank 1 updates make good sense when one seeks the optimal coefficients. On the other hand, all the kernel values involving support vectors must be stored in memory. The LaSVM direction searches are more amenable to caching strategies for kernel values.

SGD-QN Both LaSVM and SGD-QN (presented in Chapter 3) optimize SVMs for binary classification. It is interesting to compare them even if, of course, LaSVM is more general: (i) it can be used efficiently on any kind of kernel when SGD-QN is restricted to the linear case, (ii) it trains classifiers with bias terms resulting in potential higher accuracies [Keerthi *et al.*, 1999].

If we restrict to linear SVMs without bias, is it better to use LaSVM or SGD-QN? It is worth noting that, in the linear case, a smart implementation of LaSVM bypassing the kernel cache is essential to be competitive. We ran preliminary experiments (not shown in this thesis): LaSVM appears to be slightly faster than LibLinear [Hsieh *et al.*, 2008] on data sets used in Section 3.2.3 but does not outperform SGD-QN. A difference between LaSVM and SGD-QN is that LaSVM does not require fiddling with learning rates. Although this is often viewed as an advantage, we feel

that this aspect restricts the improvement opportunities and explains why SGD-QN is somewhat more efficient.

4.3 Active Selection of Training Examples

The previous section presents LaSVM as an Online Learning algorithm or as a Stochastic Optimization algorithm. In both cases, the LaSVM online iterations pick random training examples. The current section departs from this framework and investigates more refined ways to select an informative example for each iteration.

This departure is justified in the batch setup because the complete training set is available beforehand and can be searched for informative examples. It is also justified in the online setup when the continuous stream of fresh training examples is too costly to process, either because the computational requirements are too high, or because it is impractical to label all the potential training examples.

In particular, we show that selecting informative examples yields considerable speedups. Besides, training example selection can be achieved without the knowledge of the training example labels. In fact, excessive reliance on the training example labels can have very detrimental effects.

4.3.1 Example Selection Strategies

Gradient Selection

The most obvious approach consists in selecting an example k such that the Process operation results in a large increase of the dual objective function. This can be approximated by choosing the example which yields the τ-violating pair with the largest gradient. Depending on the class y_k, the Process(k) operation considers pair (k, j) or (i, k) where i and j are the indices of the examples in $\mathcal{S}$ with extreme gradients.

$$i = \arg\max_{s \in \mathcal{S}} g_s \quad \text{with } \alpha_s < B_s \ , \qquad j = \arg\min_{s \in \mathcal{S}} g_s \quad \text{with } \alpha_s > A_s$$

The corresponding gradients are $g_k - g_j$ for positive examples and $g_i - g_k$ for negative examples. Using the expression (4.2) of the gradients and the value of b and δ computed during the previous Reprocess (4.11), we can write:

$$\text{when } y_k = +1, \quad g_k - g_j = y_k\, g_k - \frac{g_i + g_j}{2} + \frac{g_i - g_j}{2} = 1 + \frac{\delta}{2} - y_k\, f(x_k)$$

$$\text{when } y_k = -1, \quad g_i - g_k = \frac{g_i + g_j}{2} + \frac{g_i - g_j}{2} + y_k\, g_k = 1 + \frac{\delta}{2} - y_k\, f(x_k)$$

This expression shows that the *Gradient Selection Criterion* simply suggests to pick the most misclassified example.

$$k_G \quad = \quad \arg\min_{k \notin \mathcal{S}} \quad y_k\, f(x_k) \tag{4.13}$$

Active Selection

Always picking the most misclassified example is reasonable when one is very confident of the training example labels. On noisy data sets, this strategy is simply going to pick mislabelled examples or examples that sit on the wrong side of the optimal decision boundary.

When training example labels are unreliable, a conservative approach chooses the example k_A that yields the strongest mini-max gradient:

$$k_A = \arg\min_{k \notin \mathcal{S}} \max_{y=\pm 1} y\, f(x_k) = \arg\min_{k \notin \mathcal{S}} |f(x_k)| \tag{4.14}$$

This *Active Selection Criterion* simply chooses the example that comes closest to the current decision boundary. Such a choice yields a gradient approximately equal to $1 + \delta/2$ regardless of the true class of the example.

Criterion (4.14) does not depend on the labels y_k. The resulting learning algorithm only uses the labels of examples that have been selected during the previous online iterations. This is related to the *Pool Based Active Learning* paradigm [Cohn *et al.*, 1990].

Early active learning literature, also known as *Experiment Design* [Fedorov, 1972], contrasts the passive learner, who observes examples (x, y), with the active learner, who constructs queries x and observes their labels y. In this setup, the active learner cannot beat the passive learner because he lacks information about the input pattern distribution [Eisenberg and Rivest, 1990]. Pool-based active learning algorithms observe the pattern distribution from a vast pool of unlabelled examples. Instead of constructing queries, they incrementally select unlabelled examples x_k and obtain their labels y_k from an oracle.

Several authors [Campbell *et al.*, 2000, Schohn and Cohn, 2000, Tong and Koller, 2000] propose incremental active learning algorithms that clearly are related to Active Selection. The initialization consists of obtaining the labels for a small random subset of examples. A SVM is trained using all the labelled examples as a training set. Then one searches the pool for the unlabelled example that comes closest to the SVM decision boundary, one obtains the label of this example, retrains the SVM and reiterates the process.

Randomized Search

Both criteria (4.13) and (4.14) suggest a search through all the training examples. This is impossible in the online setup and potentially expensive in the batch setup.

It is however possible to locate an approximate optimum by simply examining a small constant number of randomly chosen examples. The randomized search first samples M random training examples and selects the best one among these M examples. With probability $1 - \eta^M$, the value of the criterion for this example exceeds the η-quantile of the criterion for all training examples [Schölkopf and Smola, 2002, Theorem 6.33] regardless of the size of the training set. In practice this means that the best among 59 random training examples has 95% chances to belong to the best 5% examples in the training set.

Randomized search has been used in the batch setup to accelerate various machine learning algorithms [Domingo and Watanabe, 2000, Vishwanathan *et al.*, 2003, Tsang *et al.*, 2005]. In the online setup, randomized search is the only practical way to select training examples. For instance, here is a modification of the basic LaSVM algorithm to select examples using the Active Selection Criterion with Randomized Search:

Each online iteration of the above algorithm is about M times more computationally expensive that an online iteration of the basic LaSVM algorithm. Indeed one must compute the kernel expansion (2.2) for M fresh examples instead of a single one (4.2). This cost can be reduced by heuristic techniques for adapting M to the current conditions. For instance, we present experimental results where one stops collecting new examples as soon as $\mathcal{M}$ contains five examples such that $| f(x_s) | < 1 + \delta/2$.

Finally the last two paragraphs of Appendix B discuss the convergence of LaSVM with example selection according to the gradient selection criterion or the active selection criterion.

Algorithm 14 LaSVM+ Active Example Selection + Randomized Search

1: **Initialization**:
Seed $\mathcal{S}$ with a few examples of each class.
Set $\boldsymbol{\alpha} \leftarrow \mathbf{0}$ and $\boldsymbol{g} \leftarrow \mathbf{0}$.
2: **Online Iterations**:
3: Repeat a predefined number of times:
- Pick M random examples $s_1 \ldots s_M$.
- $k_t \leftarrow \underset{i=1\ldots M}{\arg\min} \mid f(x_{s_i}) \mid$
- Run Process(k_t).
- Run Reprocess once.

4: **Finishing**:
Repeat Reprocess until $\delta \leq \tau$.

The gradient selection criterion always leads to a solution of the SVM problem. On the other hand, the active selection criterion only does so when one uses the sampling method. In practice this convergence occurs very slowly. The next section presents many reasons to prefer the intermediate kernel classifiers visited by this algorithm.

4.3.2 Experiments on Example Selection for Online SVMs

This section experimentally compares the LaSVM algorithm using different example selection methods. Four different algorithms are compared:

- random example selection randomly picks the next training example among those that have not yet been Processed. This is equivalent to the plain LaSVM algorithm discussed in Section 4.2.
- gradient example selection consists in sampling 50 random training examples among those that have not yet been Processed. The sampled example with the smallest $y_k f(x_k)$ is then selected.
- active example selection consists in sampling 50 random training examples among those that have not yet been processed. The sampled example with the smallest $|f(x_k)|$ is then selected.
- autoactive example selection attempts to adaptively select the sampling size. Sampling stops as soon as 5 examples are within distance $1 + \delta/2$ of the decision boundary. The maximum sample size is 100 examples. The sampled example with the smallest $|f(x_k)|$ is then selected.

Adult Data Set

We first report experiments performed on the Adult data set. This data set provides a good indication of the relative performance of the gradient and active selection criteria under noisy conditions.

Reliable results were obtained by averaging experimental results measured for 65 random splits of the full data set into training and test sets. Paired tests indicate that test error differences of 0.25% on a single run are statistically significant at the 95% level. We conservatively estimate that average error differences of 0.05% are meaningful.

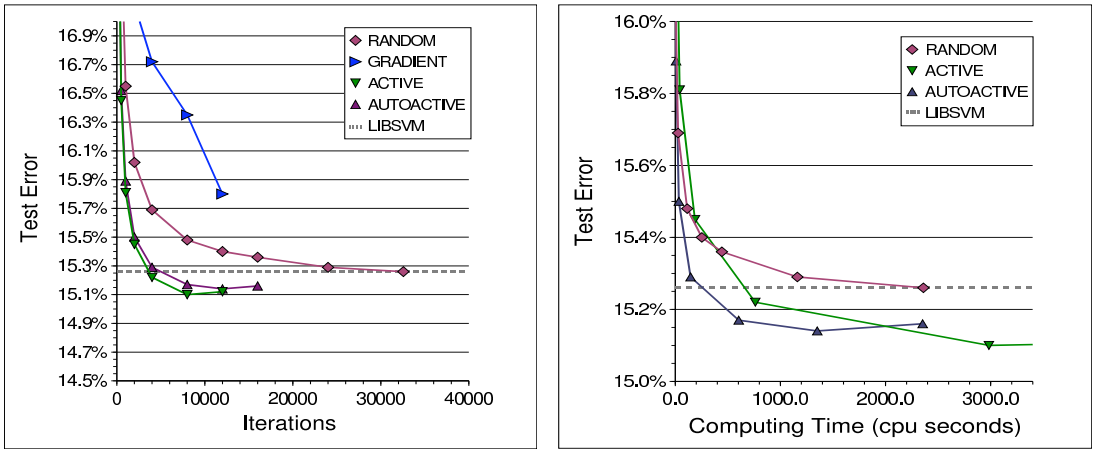


Figure 4.11: **Comparing example selection criteria on the** ADULT **data set.** Measurements were performed on 65 runs using randomly selected training sets. The graphs show the error measured on the remaining testing examples as a function of the number of iterations and the computing time. The dashed line represents the LibSVM test error under the same conditions.

Figure 4.11 reports the average error rate measured on the test set as a function of the number of online iterations (left plot) and of the average computing time (right plot). Regardless of the training example selection method, all reported results were measured after performing the LaSVM finishing step. More specifically, we run a predefined number of online iterations, save the LaSVM state, perform the finishing step, measure error rates and number of support vectors, and restore the saved LaSVM state before proceeding with more online iterations. Computing time includes the duration of the online iterations and the duration of the finishing step.

The GRADIENT example selection criterion performs very poorly on this noisy data set. A detailed analysis shows that most of the selected examples become support vectors with coefficient reaching the upper bound C. The ACTIVE and AUTOACTIVE criteria both reach smaller test error rates than those achieved by the SVM solution computed by LibSVM. The error rates then seem to increase towards the error rate of the SVM solution (left plot). We believe indeed that continued iterations of the algorithm eventually yield the SVM solution.

Figure 4.12 relates error rates and numbers of support vectors. The RANDOM LaSVM algorithm performs as expected: a single pass over all training examples replicates the accuracy and the number of support vectors of the LibSVM solution. Both the ACTIVE and AUTOACTIVE criteria yield kernel classifiers with the same accuracy and much less support vectors. For instance, the AUTOACTIVE LaSVM algorithm reaches the accuracy of the LibSVM solution using 2500 support vectors instead of 11278. Figure 4.11 (right plot) shows that this result is achieved after 150 seconds only. This is about one fifteenth of the time needed to perform a full RANDOM LaSVM epoch[6].

Both the ACTIVE LaSVM and AUTOACTIVE LaSVM algorithms exceed the LibSVM accuracy after a few iterations only. This is surprising because these algorithms only use the training labels of the few selected examples. They both outperform the LibSVM solution by using only a small subset of the available training labels.

[6]The timing results reported in figure 4.3 were measured on a faster computer.

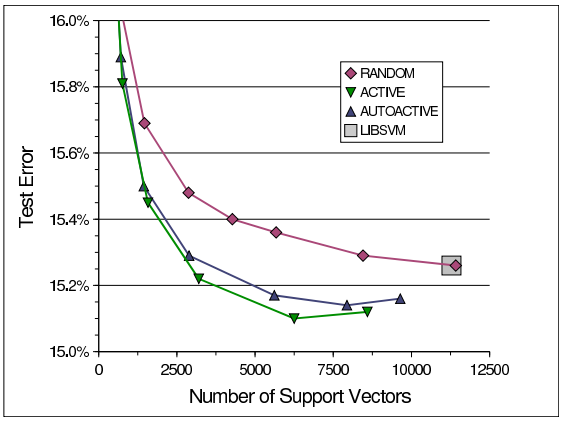


Figure 4.12: **Comparing example selection criteria on the** ADULT **data set.** Test error as a function of the number of support vectors.

MNIST Data Set

The comparatively clean MNIST data set provides a good opportunity to verify the behavior of the various example selection criteria on a problem with a much lower error rate.

Figure 4.13 compares the performance of the RANDOM, GRADIENT and ACTIVE criteria on the classification of digit "8" versus all other digits. The curves are averaged on 5 runs using different random seeds. All runs use the standard MNIST training and test sets. Both the GRADIENT and ACTIVE criteria perform similarly on this relatively clean data set. They require about as much computing time as RANDOM example selection to achieve a similar test error.

Adding ten percent label noise on the MNIST training data provides additional insight regarding the relation between noisy data and example selection criteria. Label noise was not applied to the testing set because the resulting measurement can be readily compared to test errors achieved by training SVMs without label noise. The expected test errors under similar label noise conditions can be derived from the test errors measured without label noise. Figure 4.14 shows the test errors achieved when 10% label noise is added to the training examples. The GRADIENT selection criterion causes a very chaotic convergence because it keeps selecting mislabelled training examples. The ACTIVE selection criterion is obviously undisturbed by the label noise.

Figure 4.15 summarizes error rates and number of support vectors for all noise conditions. In the presence of label noise on the training data, LibSVM yields a slightly higher test error rate, and a much larger number of support vectors. The RANDOM LaSVM algorithm replicates the LibSVM results after one epoch. Regardless of the noise conditions, the ACTIVE LaSVM algorithm reaches the accuracy and the number of support vectors of the LibSVM solution obtained with clean training data. Although we have not been able to observe it on this data set, we expect that, after a long time, the ACTIVE curve for the noisy training set converges to the accuracy and the number of support vectors achieved of the LibSVM solution obtained for the noisy data.

Online SVMs for Active Learning

The ACTIVE LaSVM algorithm implements two dramatic speedups with respect to existing active learning algorithms such as [Campbell *et al.*, 2000, Schohn and Cohn, 2000, Tong and Koller,

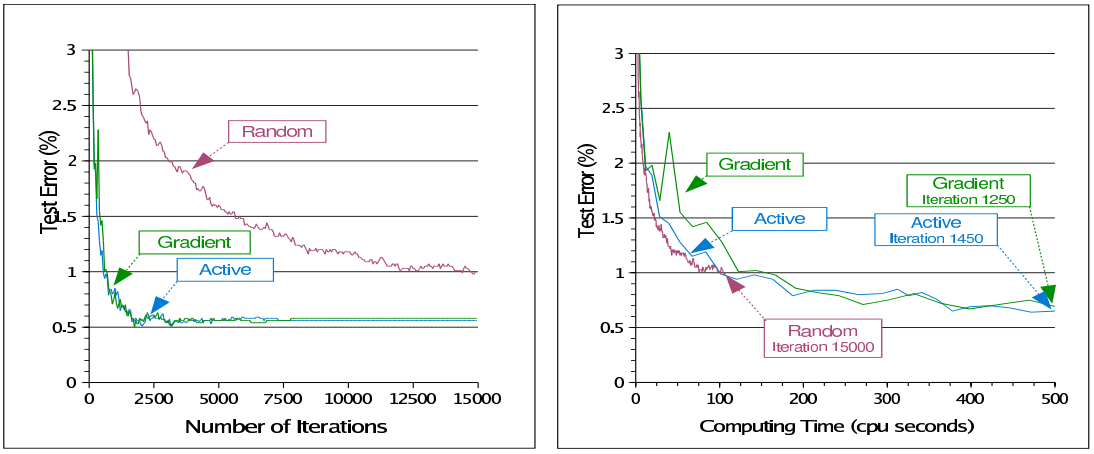


Figure 4.13: **Comparing example selection criteria on the** MNIST **data set.** (recognizing digit "8" against all other classes.) GRADIENT selection and ACTIVE selection perform similarly on this relatively noiseless task.

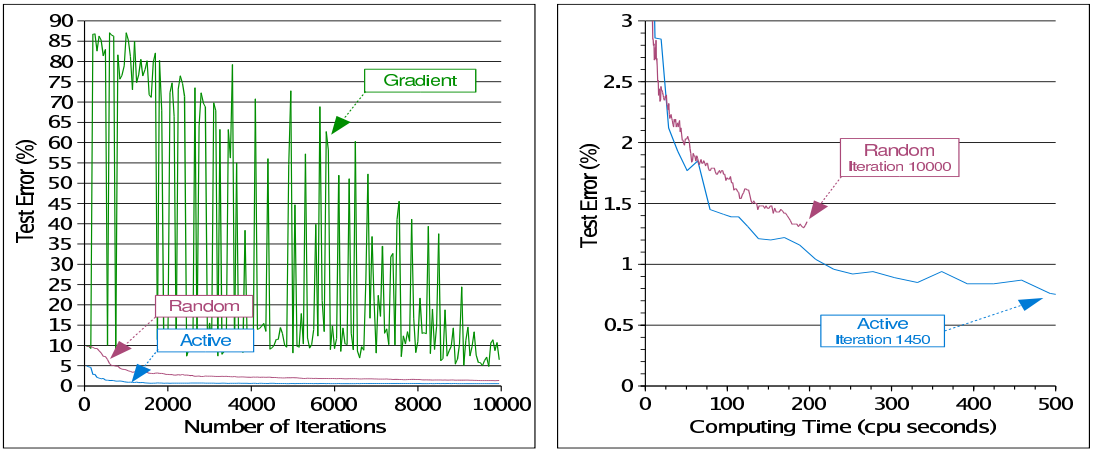


Figure 4.14: **Comparing example selection criteria on the** MNIST **data set with 10% label noise on the training examples.**

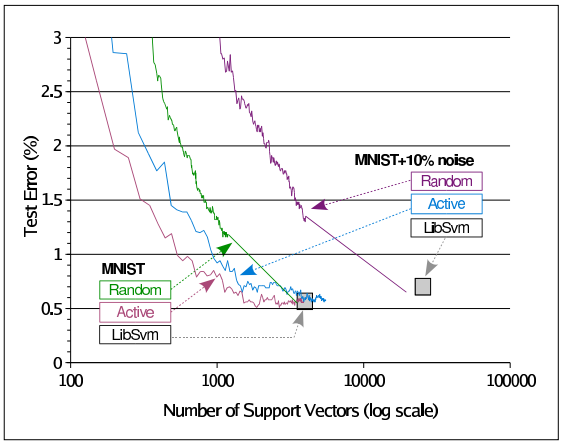


Figure 4.15: **Comparing example selection criteria on the** MNIST **data set.** ACTIVE example selection is insensitive to the artificial label noise.

2000]. First it chooses a query by sampling a small number of random examples instead of scanning all unlabelled examples. Second, it uses a single LaSVM iteration after each query instead of fully retraining the SVM.

Figure 4.16 reports experiments performed on the REUTERS and USPS data sets presented in Table 4.2. The RETRAIN ACTIVE 50 and RETRAIN ACTIVE ALL select a query from 50 or all unlabeled examples respectively, and then retrain the SVM. The SVM solver was initialized with the solution from the previous iteration. The LASVM ACTIVE 50 and LASVM ACTIVE ALL do not retrain the SVM, but instead make a single LaSVM iteration for each new labeled example.

All the active learning methods performed approximately the same, and were superior to random selection. Using LaSVM iterations instead of retraining causes no loss of accuracy. Sampling $M = 50$ examples instead of searching all examples only causes a minor loss of accuracy when the number of labeled examples is very small. Yet the speedups are very significant: for 500 queried labels on the Reuters data set, the RETRAIN ACTIVE ALL, LASVM ACTIVE ALL, and LASVM ACTIVE 50 algorithms took 917 seconds, 99 seconds, and 9.6 seconds respectively.

4.3.3 Discussion

Practical Significance

As we discussed in Chapter 1, data set sizes are quickly outgrowing the computing power of our calculators. One possible avenue consists in harnessing the computing power of multiple computers [Graf *et al.*, 2005]. In this thesis, we are rather seeking learning algorithms with low complexity.

When we have access to an abundant source of training examples, the simple way to reduce the complexity of a learning algorithm consists of picking a random subset of training examples and running a regular training algorithm on this subset. Unfortunately this approach renounces the more accurate models that the large training set could afford. This is why we say, by reference to statistical efficiency, that an *efficient* learning algorithm should at least pay a brief look at every training example.

The LaSVM algorithm is very attractive because it yields competitive results after a single epoch. This is very important in practice because modern data storage devices are most effective

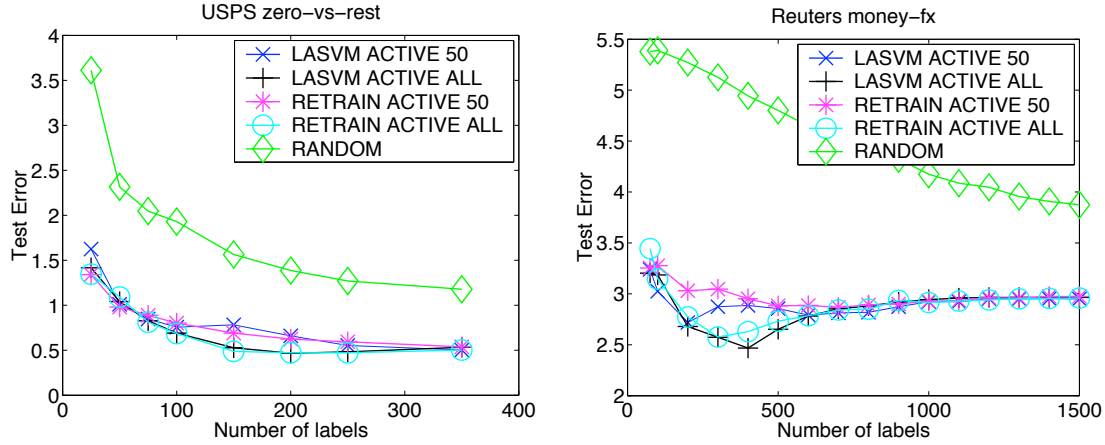


Figure 4.16: **Comparing active learning methods on the** USPS **and** Reuters **data sets.** Results are averaged on 10 random choices of training and test sets. Using LaSVM iterations instead of *retrain*ing causes no loss of accuracy. Sampling $M = 50$ examples instead of searching all examples only causes a minor loss of accuracy when the number of labeled examples is very small.

when the data is accessed sequentially. Active Selection of the LaSVM training examples brings two additional benefits for practical applications: (a) it achieves equivalent performances with significantly less support vectors, and (b) the search for informative examples is an obviously parallel task.

Informative Examples and Support Vectors

By suggesting that all examples should not be given equal attention, we first state that all training examples are not equally informative. This question has been asked and answered in various contexts [Fedorov, 1972, Cohn *et al.*, 1990, MacKay, 1992]. We also ask whether these differences can be exploited to reduce the computational requirements of learning algorithms. Our work answers this question by proposing algorithms that exploit these differences and achieve very competitive performances.

Kernel classifiers in general distinguish the few training examples named support vectors. Kernel classifier algorithms usually maintain an active set of potential support vectors and work by iterations. Their computing requirements are readily associated with the training examples that belong to the active set. Adding a training example to the active set increases the computing time associated with each subsequent iteration because they will require additional kernel computations involving this new support vector. Removing a training example from the active set reduces the cost of each subsequent iteration. However it is unclear how such changes affect the number of subsequent iterations needed to reach a satisfactory performance level.

Online kernel algorithms, such as kernel perceptrons usually produce different classifiers when given different sequences of training examples. Section 4.2 proposes an online kernel algorithm that converges to the SVM solution after many epochs. The final set of support vectors is intrinsically defined by the SVM QP problem, regardless of the path followed by the online learning process. Intrinsic support vectors provide a benchmark to evaluate the impact of changes in the active set of current support vectors. Augmenting the active set with an example that is not an intrinsic support vector moderately increases the cost of each iteration without clear

benefits. Discarding an example that is an intrinsic support vector incurs a much higher cost. Additional iterations will be necessary to recapture the missing support vector.

Nothing guarantees however that the most informative examples are the support vectors of the SVM solution. [Bakır *et al.*, 2005] interpret Steinwart's theorem [Steinwart, 2004] as an indication that the number of SVM support vectors is asymptotically driven by the examples located on the wrong side of the optimal decision boundary. Although such outliers might play a useful role in the construction of a decision boundary, it seems unwise to give them the bulk of the available computing time. Section 4.3 adds explicit example selection criteria to LaSVM. The Gradient Selection Criterion selects the example most likely to cause a large increase of the SVM objective function. Experiments show that it prefers outliers over honest examples. The Active Selection Criterion bypasses the problem by choosing examples without regard to their labels. Experiments show that it leads to competitive test error rates after a shorter time, with less support vectors, and using only the labels of a small fraction of the examples.

Theoretical Questions

Appendix B provides a comprehensive analysis of the convergence of the algorithms discussed in this contribution. Such convergence results are useful but limited in scope. This section underlines some aspects of this work that would vastly benefit from a deeper theoretical understanding.

- Test error rates are sometimes improved by active example selection. In fact this effect has already been observed in the active learning setups [Schohn and Cohn, 2000]. This small improvement is difficult to exploit in practice because it requires very sensitive early stopping criteria. Yet it demands an explanation because it seems that one gets a better performance by using less information. There are three potential explanations: (*i*) active selection works well on unbalanced data sets because it tends to pick equal number of examples of each class [Schohn and Cohn, 2000], (*ii*) active selection improves the SVM loss function because it discards distant outliers, (*iii*) active selection leads to more sparse kernel expansions with better generalization abilities [Cesa-Bianchi *et al.*, 2005]. These three explanations may be related and some recent work actually explore them using LaSVM [Ertekin *et al.*, 2007a, Ertekin *et al.*, 2007b].
- We know that the number of SVM support vectors scales linearly with the number of examples [Steinwart, 2004]. Empirical evidence suggests that active example selection yields transitory kernel classifiers that achieve low error rates with much less support vectors. What is the scaling law for this new number of support vectors?

We have presented empirical evidence suggesting that a single epoch of the LaSVM algorithm yields misclassification rates comparable with a SVM. We also know that LaSVM exactly reaches the SVM solution after a sufficient number of epochs. For well designed online learning algorithms based on Stochastic Gradient Descent, there exist theoretical results estimating the expected difference between the first epoch test error and the many epoch test error (see Theorem 4 in Section 3.1.1). In the next section, we provide original theoretical guarantees for the online LaSVM (and incremental algorithms). Indeed, using a new duality lemma, we demonstrate that using a fixed number of Reprocess operations allows to track the SVMs optimum on the course of learning.

4.4 Tracking Guarantees for Online SVMs

Standard online learning algorithms, like the perceptron, passive-aggressive algorithms, or stochastic gradient descent, perform a single parameter update after seeing each new example. As we

have already discussed, these are faster than batch algorithms that optimize a global cost function on the whole training set but have usually a significantly worse test performance. Running several pass over a fixed training set can yet often turn them into computationally efficient learning algorithms. They become as accurate in test as batch optimizers and are generally still competitive in terms of training time. However they are no longer online and this involves drawbacks. In particular, as mentioned in the discussion on kernel cache usage of Section 4.2.5, caching requirements of an algorithm increase a lot when it runs multiple passes.

We have shown in Sections 4.2 and 4.3 that LaSVM does not require to be looped several times over the training set to reach good performances. On various learning tasks it reaches a test accuracy nearly as good as the final solution and a dual objective value close to the optimum, after *a single epoch* over the training set. This has empirically demonstrated the rewarding influence of the addition of a limited number of Reprocess steps. In this section we now attempt to give theoretical insights of this useful impact.

We study SVMs algorithms that do not try to reach the optimum of the SVMs QP at each time index t – as usually do incremental algorithms [Cauwenberghs and Poggio, 2001] – but *strive to track the sequence of optima* with a predefined tolerance. Our analysis shows that adequately designed algorithms can track the successive optima by performing a constant number of iterations for each additional example (similar in spirit than Reprocess operations). This results in an optimality guarantee that can be obtained with no extra-computation. The total number of required iterations grows linearly with the number of examples and as for the best algorithms for computing approximate SVMs [Joachims, 2006, Shalev-Shwartz *et al.*, 2007].

We first describe our analysis setup (Section 4.4.1) and give a useful duality lemma (Section 4.4.2). Then we present and analyze two approximate incremental SVM algorithms (Section 4.4.3) and conclude with a discussion on how this translates to LaSVM (Section 4.4.4).

4.4.1 Analysis Setup

We consider the following online setup. Examples arrive as a stream of examples $(x_i, y_i)_{i\geq 1}$ with instances x_i verifying $\|x_i\| \leq 1$ and with labels $y_i = \pm 1$. We consider discriminant functions of the form $f(x) = \langle w, x\rangle$ (we use no bias). Throughout this section, we only use the linear kernel function $k(x, \bar{x}) = \langle x, \bar{x}\rangle$ but all the results we demonstrate could be translated to any general kernel function.

As usual, we let $P_t(w)$ be the primal cost function restricted to the set S_t containing the first t examples,

$$P_t(w) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{t} \max(0, 1 - y_i \langle w, x_i\rangle) \tag{4.15}$$

and let $D_t(\boldsymbol{\alpha})$ be the associated dual objective function

$$D_t(\boldsymbol{\alpha}) = \sum_{i=1}^{t} \alpha_i - \frac{1}{2}\sum_{i,j\leq t} y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle \quad \text{with} \quad \forall i = 1, \dots t, \;\; 0 \leq \alpha_i \leq C\,. \tag{4.16}$$

We employ here the standard dual formulation of SVMs, which is slightly different from the one previously used in this chapter, because this eases notations. Of course, the two forms are equivalent and lead to the same final vector w.

If $\boldsymbol{\alpha}^*$ maximizes D_t, it is well known that[7] $w(\boldsymbol{\alpha}^*) = \sum_{i=1}^{t} \alpha_i^* y_i x_i$ minimizes P_t, and

$$D_t^* = D_t(\boldsymbol{\alpha}^*) = \max_{\boldsymbol{\alpha} \in [0,C]^t} D_t(\boldsymbol{\alpha}) \;=\; \min_{w} P_t(w) = P_t(w(\boldsymbol{\alpha}^*)) = P_t^* \,.$$

Dual coordinate ascent is a simple procedure to maximize D_t. It is similar to Sequential Direction Search presented in Section 2.1.2 but simpler because, as we removed the bias term, there is no equality constraint in the dual anymore. Let $(\boldsymbol{e}_1 \dots \boldsymbol{e}_t)$ be the canonical basis of $\mathbb{R}^t$. Starting from a dual parameter vector $\boldsymbol{\alpha}^k \in [0,C]^t$, each dual coordinate ascent iteration picks a search direction $\boldsymbol{e}_{\sigma(k)}$ and outputs a new dual parameter vector $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + a^* \boldsymbol{e}_{\sigma(k)}$ with a^* chosen to maximize $D(\boldsymbol{\alpha}^{k+1})$ subject to $\boldsymbol{\alpha}^{k+1} \in [0,C]^t$. A simple derivation shows that

$$a^* = \max\big(-\alpha^k_{\sigma(k)}\,,\, \min\big(C - \alpha^k_{\sigma(k)}, \frac{g_{\sigma(k)}(\boldsymbol{\alpha}^k)}{\big\|x_{\sigma(k)}\big\|^2}\big)\big) \quad \text{with} \quad g_i(\boldsymbol{\alpha}) = 1 - y_i \left\langle w(\boldsymbol{\alpha}), x_i \right\rangle \,. \tag{4.17}$$

An approximate minimizer of the primal cost D can therefore be obtained by choosing a suitable starting value $\boldsymbol{\alpha}^0$, performing an adequate number K of successive dual coordinate ascent iterations and outputting $w(\boldsymbol{\alpha}^K)$. The convergence and the efficiency of this procedure depends on the *scheduling policy* used to chose the successive search direction $\boldsymbol{e}_{\sigma(k)}$ at each step.

4.4.2 Duality Lemma

The following lemma is interesting because it connects the two quantities of interest: the duality gap (i.e. the difference between the primal and the corresponding dual costs), which measures the accuracy of the solution, and the expected effect of the next coordinate ascent iteration.

Lemma 5 *Let $t \geq 1$, $\max_{i=1..t} \|x_i\| \leq 1$, and $\boldsymbol{\alpha} \in [0,C]^t$. Then:*

$$\frac{P_t(w(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha})}{Ct} \leq \mu\left(\mathop{\mathbb{E}}_{i \sim U(t)} \Delta_{t,i}(\boldsymbol{\alpha})\right)$$

where $\mu(x) = \sqrt{2x} + x/C$, $U(t)$ denotes the uniform distribution over $\{1...t\}$, and

$$\Delta_{t,i}(\boldsymbol{\alpha}) = \max_{a \in [-\alpha_i, C-\alpha_i]} [D_t(\boldsymbol{\alpha} + a\boldsymbol{e}_i) - D_t(\boldsymbol{\alpha})]$$

A bound on the gap is of course a bound on both the primal $P_t(w(\boldsymbol{\alpha})) - P_t^*$ and dual $D_t(\boldsymbol{\alpha}) - D^*$ suboptimalities. The left hand side denominator Ct makes sense because it normalizes the loss in the expression of the primal (4.15).

Proof The result follows from elementary arguments regarding $\Delta_{t,i}(\boldsymbol{\alpha})$ and the duality gap

$$G(\boldsymbol{\alpha}) = P_t(w(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha}) = \|w(\boldsymbol{\alpha})\|^2 + C \sum_{i=1}^{t} \max(0, g_i(\boldsymbol{\alpha})) - \sum_{i=1}^{t} \alpha_i$$

Recalling $\|w(\boldsymbol{\alpha})\|^2 = \sum_{i=1}^{t} y_i \alpha_i \left\langle w(\boldsymbol{\alpha}), x_i \right\rangle = \sum_{i=1}^{t} \alpha_i \big(1 - g_i(\boldsymbol{\alpha})\big)$, we obtain the identity

$$G(\boldsymbol{\alpha}) = \sum_{i=1}^{t} \max\left[(C - \alpha_i) g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] \,. \tag{4.18}$$

[7] In this section, we denote the parameter vector $w(\boldsymbol{\alpha})$ to make explicit its dependency on the vector $\boldsymbol{\alpha}$.

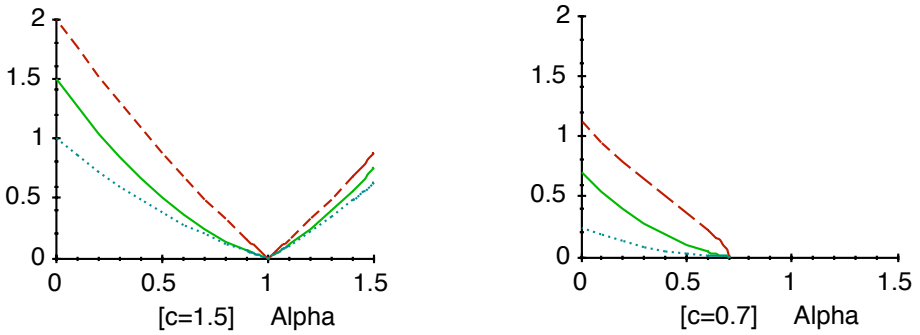


Figure 4.17: **Duality lemma with a single example** $x_1 = 1$**,** $y_1 = 1$**.** The figures compare the gap $P_t - D_t$ (continuous green curve), the bound (dashed red curve), and the primal suboptimality $P_t - P_t^*$ (dotted blue curve) as a function of α_1. The left plot shows a free support vector ($C = 1.5$, $\alpha_1^* = 1$). The right plot shows a support vector at bound ($C = 0.7$, $\alpha_1^* = C$).

We now turn our attention to quantity $\Delta_{t,i}(\boldsymbol{\alpha})$. Equation (4.17) shows that a^* has always the same sign as $g_i(\boldsymbol{\alpha})$ and $|a^*| \leq |\ g_i|/\|x_i\|^2$. Since $D_t(\boldsymbol{\alpha} + a\boldsymbol{e}_i) - D_t(\boldsymbol{\alpha}) = a\left(g_i(\boldsymbol{\alpha}) - a/2\,\|x_i\|^2\right)$,

$$\frac{1}{2}|a^*||g_i(\boldsymbol{\alpha})| \leq \Delta_{t,i}(\boldsymbol{\alpha}) = |a^*||g_i(\boldsymbol{\alpha})| - \frac{1}{2}\|x_i\|^2|a^*|^2\,. \tag{4.19}$$

To use this result in equation (4.18), we fix some index i and consider two cases:

1. If $|a^*| \geq |g_i(\boldsymbol{\alpha})|$, then, using equation (4.19), we have $|g_i(\boldsymbol{\alpha})| \leq \sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})}$, and thus

$$\max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] \leq C|g_i(\boldsymbol{\alpha})| \leq C\sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})} \leq C\mu(\Delta_{t,i}(\boldsymbol{\alpha}))\,.$$

2. If $|a^*| < |g_i(\boldsymbol{\alpha})|$, then, given (4.17) and the assumption $\|x_i\|^2 \leq 1$, $\alpha_i + a^*$ has necessarily reached a bound. Since a^* and $g_i(\boldsymbol{\alpha})$ have the same sign, it means that if $g_i(\boldsymbol{\alpha}) \leq 0$, then $a^* = -\alpha_i$, and $a^* = C - \alpha_i$ otherwise. This implies

$$\max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] = |a^*||g_i(\boldsymbol{\alpha})| = \Delta_{t,i}(\boldsymbol{\alpha}) + \frac{1}{2}\|x_i\|^2|a^*|^2\,.$$

   In order to obtain a bound involving only $\Delta_{t,i}(\boldsymbol{\alpha})$, we need to bound the last term of the last equation. Since we are in the case $|a^*| < |g_i(\boldsymbol{\alpha})|$, the left-hand side of equation (4.19) gives us $|a^*| \leq \sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})}$. Moreover, since $\|x_i\|^2 \leq 1$, we have $\frac{1}{2}\|x_i\|^2|a^*| \leq \frac{1}{2}C$ and

$$\max\left[(C-\alpha_i)g_i(\boldsymbol{\alpha}), -\alpha_i g_i(\boldsymbol{\alpha})\right] \leq \Delta_{t,i}(\boldsymbol{\alpha}) + \frac{1}{2}C\sqrt{2\Delta_{t,i}(\boldsymbol{\alpha})} \leq C\mu(\Delta_{t,i}(\boldsymbol{\alpha}))\,.$$

Putting points 1 and 2 in equation (4.18), and using the concavity of μ, we obtain the desired result:

$$\frac{P_t(w(\boldsymbol{\alpha})) - D_t(\boldsymbol{\alpha})}{Ct} \leq \frac{1}{t}\sum_{i=1}^{t}\mu(\Delta_{t,i}(\boldsymbol{\alpha})) \leq \mu\left(\mathop{\mathbb{E}}_{i\sim U(t)}\Delta_{t,i}(\boldsymbol{\alpha})\right)\,.$$

□

To ascertain the quality of this bound, consider how a SVM with a single scalar example $x_1 = 1$, $y_1 = 1$ illustrates the two cases of the proof. The left plot in Figure 4.17 shows a situation where the optimum uses the example as a free support vector, that is, case 1 in the proof. The lack of a vertical tangent near the optimum $\alpha_1 = 1$ confirms the square root behavior of $\mu(x)$ when x approaches zero. The right plot shows a situation of a bounded support vector, that is, case 2 in the proof. The bound is much looser when α_i approaches C. However this is less important because coordinate ascent iterations usually set such coefficient to C at once. The bound is then exact.

4.4.3 Algorithms and Analysis

The Analysis Technique

Let us illustrate the analysis technique on the dual coordinate ascent algorithm outlined in Section 4.4.1 running on a fixed training set with t examples. Assume the successive search directions are picked randomly. We can easily copy the collapsing sum method of [Shalev-Shwartz and Singer, 2007b].

Let $\mathcal{F}_k$ represent all the successive search directions $\boldsymbol{e}_{\sigma(i)}$, $i < k$. We can rewrite Lemma 5 as

$$\forall k \quad \frac{P_t(w(\boldsymbol{\alpha}^k)) - D_t(\boldsymbol{\alpha}^k)}{Ct} \leq \mu\left(\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\middle|\,\mathcal{F}_k\right]\right) .$$

Taking the expectation, averaging over all k, and using twice Jensen's inequality,

$$\begin{aligned}\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[\frac{P_t(w(\boldsymbol{\alpha}^k)) - D_t(\boldsymbol{\alpha}^k)}{Ct}\right] &\leq \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left[\mu\left(\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\middle|\,\mathcal{F}_k\right]\right)\right] \\ \leq \mu\left(\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} D_t(\boldsymbol{\alpha}^{k+1}) - D_t(\boldsymbol{\alpha}^k)\right]\right) &\leq \mu\left(\frac{\mathbb{E}\left[D_t(\boldsymbol{\alpha}^{K+1}) - D_t(\boldsymbol{\alpha}^1)\right]}{K}\right) \leq \mu\left(\frac{\mathbb{E}\, D_t^*}{K}\right) .\end{aligned}$$

Since the gap bounds both the primal and dual suboptimality, we obtain a dual convergence bound

$$\mathbb{E}\left[\frac{D_T^* - D_t(\boldsymbol{\alpha}^K)}{Ct}\right] \leq \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{D_t^* - D_t(\boldsymbol{\alpha}^k)}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\, D_t^*}{K}\right) ,$$

and a somehow less attractive primal convergence bound

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{P_t(w(\boldsymbol{\alpha}^k)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\, D_t^*}{K}\right) .$$

These bounds are different because each iteration increases the value of the dual objective, but does not necessarily reduce the value of the primal cost. However it is easy to obtain a nicer primal convergence bound by considering an averaged algorithm. Let $\bar{\boldsymbol{\alpha}}^K = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{\alpha}_k$. Thanks to the convexity of the primal cost, we can write

$$\mathbb{E}\left[\frac{P_t(w(\bar{\boldsymbol{\alpha}}^K)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\, D_t^*}{K}\right) .$$

In practice, this averaging operation is dubious because it ruins the sparsity of the dual parameter vector $\boldsymbol{\alpha}$. However it yields bounds that are easier to interpret, albeit not fundamentally different.

Tracking Inequality for a Simple Algorithm

We now return to an incremental setup. Assume a teacher provides a new example (x_t, y_t) at each time step. We seek to compute a sequence of classifiers w_t that tracks $P_t^* = \min_w P_t(w)$ with a predefined accuracy.

Algorithm 15 Simple Averaged Tracking Algorithm

1: **input**: stream of examples $(x_i, y_i)_{i \geq 1}$, number of iterations $K \geq 1$ at each time index.
2: $\forall i, \alpha_i \leftarrow 0$, $t \leftarrow 1$
3: Pick an example (x_t, y_t)
4: Set $\alpha_t \leftarrow 0$
5: **for** $k = 1, \dots K$ **do**
6: Pick i randomly in $\{1, ..., t\}$
7: Set $\alpha_i \leftarrow \alpha_i + \max\left(-\alpha_i, \min\left(C - \alpha_i, \frac{g_i(\boldsymbol{\alpha})}{\|x_i\|^2}\right)\right)$
8: Set $\bar{\boldsymbol{\alpha}}^t \leftarrow \frac{k-1}{k}\bar{\boldsymbol{\alpha}}^t + \frac{1}{k}\boldsymbol{\alpha}$
9: **end for**
10: **output** classifier $w^t = w(\bar{\boldsymbol{\alpha}}^t)$
11: $t \leftarrow t + 1$
12: Return to step 3.

After receiving each new example (x_t, y_t), Algorithm 15 performs a predefined number K of dual coordinate ascent iterations on randomly picked coefficients associated with the currently known examples.

Theorem 6 *Let $w(\bar{\boldsymbol{\alpha}}^t)$ be the sequence of classifiers output by Algorithm 15. Assume furthermore that $\max_t \|x_t\| \leq 1$. Then, for any $T \geq 1$, we have*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\frac{P_t(w(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\, D_T^*}{KT}\right)$$

where $\mu(x) = \sqrt{2x} + \frac{x}{C}$. Moreover, the number of dual coordinate ascent performed by the algorithm after seeing T examples is exactly KT.

Theorem 6 does not bound the primal suboptimality at each time index. However, since all these primal suboptimalities are positive, the theorem guarantees that an upper bound of the excess misclassification error, $(P_t - P_t^*)/Ct$, will be bounded *on average*. This weaker guarantee comes with a considerable computational benefit: instead of computing costly stopping criteria, we can blindly perform K iterations after each new example and know that the guarantee holds.

The proof of the theorem follows the schema of the previous section: setup the collapsing sum of dual objective values; use Jensen's inequality to distribute the function μ and the expectations on each term; apply the lemma; and regroup the like terms on the left hand side.

Expectations in the theorem can have two interpretations. In the simplest setup, the teacher fixes the sequence of examples before the execution of the algorithm. Expectations are then taken solely with respect to the successive random choices of coordinate ascent directions. In a more general setup, the teacher follows an unspecified causal policy. At each time index t, he can use past values of the algorithm variables to choose the next example (x_t, y_t): this corresponds to an active learning setup. The sequence of examples becomes a random variable. Expectations are then taken with respect to both the random search directions and the sequence of examples. The proof is identical in both cases.

Tracking Inequality for a Process/Reprocess Algorithm

Algorithm 16 is inspired by the Process/Reprocess principle of the Huller and LaSVM (presented in Sections 4.1 and 4.2). Before performing K dual coordinate ascent iterations on coefficients associated with examples randomly picked among the currently known examples, this algorithm performs an additional iteration on the coefficient associated with the new example (compare lines 4 in both algorithms.)

Algorithm 16 Averaged Tracking Algorithm with Process/Reprocess

1: **input**: stream of examples $(x_i, y_i)_{i\geq 1}$, number of iterations $K \geq 1$ at each time index.
2: $\forall i, \alpha_i \leftarrow 0$, $t \leftarrow 1$
3: Pick an example (x_t, y_t)
4: Set $\alpha_t \leftarrow \max\left(0, \min\left(C, \frac{g_t(\boldsymbol{\alpha})}{\|x_t\|^2}\right)\right)$ ▷ i.e. perform Process
5: **for** $k = 1, \dots K$ **do**
6: Pick i randomly in $\{1, ..., t\}$
7: Set $\alpha_i \leftarrow \alpha_i + \max\left(-\alpha_i, \min\left(C - \alpha_i, \frac{g_i(\boldsymbol{\alpha})}{\|x_i\|^2}\right)\right)$ ▷ i.e. perform K Reprocess
8: Set $\bar{\boldsymbol{\alpha}}^t \leftarrow \frac{k-1}{k}\bar{\boldsymbol{\alpha}}^t + \frac{1}{k}\boldsymbol{\alpha}$
9: **end for**
10: **output** classifier $w^t = w(\bar{\boldsymbol{\alpha}}^t)$
11: $t \leftarrow t + 1$
12: Return to step 3.

Theorem 7 *Let $w(\bar{\boldsymbol{\alpha}}^t)$ be the sequence of classifiers output by Algorithm 16. Let $\boldsymbol{\alpha}^t$ denote the successive value taken by variable $\boldsymbol{\alpha}$ before each execution of line 4 of Algorithm 16. Assume furthermore that $\max_t \|x_t\| \leq 1$. Then, for any $T \geq 1$, we have*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\frac{P_t(w(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \mu\left(\frac{\mathbb{E}\left[D_T^* - \delta_T\right]}{KT}\right)$$

where $\mu(x) = \sqrt{2x} + \frac{x}{C}$ and $\delta_T = \sum_{t=1}^{T}\Delta_{t,t}(\boldsymbol{\alpha}^t)$ is the cumulated dual increase during the Process *operations. Moreover, the number of elementary optimization steps performed by the algorithm after seeing t examples is exactly $(K+1)t$.*

The proof of the theorem is similar to the proof of Theorem 6 except that terms of the collapsing sum corresponding to the Process operations (line 4 in the algorithm) are collected in quantity δ_T.

Adding this Process operation gives the bound $\mu\left(\mathbb{E}\left[D_T^* - \bar{\delta}_T\right]/KT\right)$ instead of $\mu\left(\mathbb{E}\left[D_T^*\right]/KT\right)$. Since the quantity δ_T is related to the online loss incurred by the online algorithm [Shalev-Shwartz and Singer, 2007b], $\delta_T = \Omega(T)$ unless the all training examples received after a given time index are separable. Under this condition, the Process operation saves a multiplicative factor on the number K of Reprocess operations necessary to reach a predefined accuracy. Although we cannot give a precise value for δ_T, we can claim that Algorithm 16, implementing a sort of Process/Reprocess strategy, should perform significantly better than Algorithm 15 in practice.

Rough Comparisons

Since $D_T^* - \delta_T \leq D_T^* \leq P_T(\mathbf{0}) = CT$ the following corollary can be derived from the theorems.

Corollary 8 *Under the assumptions of Theorems 6 and 7, let* $4C \geq \epsilon \geq 0$. *When* $K = \lceil \frac{8C}{\epsilon^2} \rceil$, *both Algorithms 15 and 16 satisfy* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \frac{P_t(w(\bar{\boldsymbol{\alpha}}^t)) - P_t^*}{Ct}\right] \leq \epsilon$.

The total number n of iterations therefore scales like T/ϵ^2 where T is the number of examples. Since the cost of each iteration depends on details of the algorithm (see Section 4.4.4), let us assume, as a first approximation, that the cost of each iteration is proportional to either the number of support vectors, or, in the case of linear kernels, on the effective dimension of the patterns. The results we report here are then comparable to the bounds reported in [Tsochantaridis *et al.*, 2005, Joachims, 2006, Franc and Sonnenburg, 2008]. Improved bounds for generic bundle methods [Smola *et al.*, 2008] are difficult to compare because their successive iterations solve increasingly complex optimization problems. Bounds for stochastic gradient algorithms [Shalev-Shwartz *et al.*, 2007] also scale linearly with the number T of examples[8] but offer better scaling in $1/\epsilon$.

The next section show how these orders of magnitude can be improved on concrete cases.

4.4.4 Application to LaSVM

Transfer from the study-models 15 and 16 to real algorithms like LaSVM involves using some algorithmic tricks that can procure performance gains and are therefore crucial on practical applications.

Detecting Ascent Directions that Do Nothing

Algorithms 15 and 16 select dual coordinate ascent directions randomly. As a consequence, most of these coordinate ascent iterations have no effect because the selected coefficient α_i cannot be improved. This happens when $\alpha_i = 0$, $g_i(\boldsymbol{\alpha}) \leq 0$ or when $\alpha_i = C$, $g_i(\boldsymbol{\alpha}) \geq 0$. Assume we have an efficient way to detect that the current coordinate ascent direction corresponds to one of these case. We can then simply shortcut the coordinate ascent iteration because we know that it does nothing.[9]

Given a training set of t training examples, let $n_0(\boldsymbol{\alpha})$ and $n_C(\boldsymbol{\alpha})$ be the number of examples falling into these two cases. These numbers approach $n_0(\boldsymbol{\alpha}_t^*)$ and $n_C(\boldsymbol{\alpha}_t^*)$ when $\boldsymbol{\alpha}$ approaches the SVM solution $\boldsymbol{\alpha}_t^*$. Provided that C decreases with an appropriate rate to ensure consistency, [Steinwart, 2004] has famously proved that the total number of support vectors $t - n_0(\boldsymbol{\alpha}_t^*)$ scales linearly with the total number t of examples. But his work also shows that the number of support vectors is dominated by the number $n_C(\boldsymbol{\alpha}_t^*)$ of margins violators. Therefore the fraction $(n_0(\boldsymbol{\alpha}) + n_C(\boldsymbol{\alpha}))/t$ of useless coordinate ascent iterations tends to 1 when t increases and the algorithm converges. Avoiding these operations would therefore improve the asymptotical behavior of the algorithm.

In order to test whether a coordinate ascent iteration along direction $\boldsymbol{e}_i$ is useless, we need to obtain the quantity $g_i(\boldsymbol{\alpha})$. The traditional solution in SVM solvers (and used in LaSVM) consists in allocating new variables g_i that always contain quantity $g_i(\boldsymbol{\alpha})$. Whenever a coefficient of $\boldsymbol{\alpha}$ changes, updating all the g_i variables requires a time proportional to the total number t of examples. Since this only happens when an actual update takes place, the amortized cost of a coordinate ascent iteration is then proportional to $t - n_0(\boldsymbol{\alpha}) - n_C(\boldsymbol{\alpha})$ which grows slower than t. In comparison, the direct computation of $g_i(\boldsymbol{\alpha})$ with nonlinear kernels is proportional to the

[8] [Shalev-Shwartz *et al.*, 2007] report a bound in $\tilde{O}\left(\frac{1}{\lambda\epsilon}\right)$. Their λ is $1/CT$ in our setup.

[9] It is easy to postpone the averaging operation (line 8 in the algorithms) until an actual update of $\boldsymbol{\alpha}$ takes place. We just have to count how many averaging operations are pending in order to include $\boldsymbol{\alpha}$ into the average with the appropriate weight.

number of support vectors $t - n_0(\boldsymbol{\alpha})$ which grows like t. Storing some gradient derivatives brings up non-trivial computational benefits.

Tracking Inequalities for Online LaSVM

We can even do better if we are willing to accept a weaker guarantee. Assume the teacher hands us the examples (x_t, y_t) by performing multiple sequential passes over a finite training set of T examples. We run a variant of Algorithm 16 with the following modifications:

i) we maintain variables g_i representing $g_i(\boldsymbol{\alpha})$ for only those i such that $\alpha_i > 0$,

ii) we shortcut line 7 in Algorithm 16 whenever $\alpha_i = 0$ or $\alpha_i = C\ ,\ g_i \geq 0$.

This algorithm can be viewed as a randomized variant of LaSVM. Updating the g_i is now proportional to the number of support vectors instead of the total number of examples. This brings very positive effects on the memory requirements of the kernel cache of LaSVM.

On the other hand, this modified algorithm can shortcut a coordinate ascent with $\alpha_i = 0$ that would actually do something because $g_i(\boldsymbol{\alpha}) > 0$. Yet we can carry out the analysis of Theorem 7 using a simple trick: whenever we shortcut a coordinate ascent iteration that would have updated the coefficient α_i, we simply remove the corresponding example from our current training set S_t. This removal is an artifice of the analysis that allows us to use lemma 5. Nothing changes in the algorithm since this situation only happens when $\alpha_i = 0$. As a result, the left hand side of the bound involves the average primal suboptimality on a sequence of training sets that is no longer strictly increasing. Examples removed in this way will reenter the training set during the next pass over the training set. We know that such algorithms converge (see Appendix B), so successive training sets S_t will eventually encompass all the T examples.

Experiments of the previous sections show that such removals are relatively rare. Hence, this viewpoint casts a useful light about the behavior of the Huller and LaSVM. After the first pass over the training set, the guarantee encompasses almost all the training examples and we can expect a performance close to that of the true SVM. After a couple additional passes, the removed examples have reentered the training sets S_t and the guarantee suggests that we closely match the true SVM performance. This is exactly what we experimentally observe in Sections 4.1.3 and 4.2.5.

Extension to Active Learning

Theorems 6 and 7 make very little assumptions about the teacher's policy. They state that the algorithm will track the sequence of SVM solutions after a number of coordinate ascent iterations that is independent on the quality of the teacher.

Let us assume that the teacher has T examples and chooses a presentation order π beforehand. Following [Bengio *et al.*, 2009], we call such a presentation order a *curriculum*. Algorithm 15, for instance, will perform exactly KT coordinate ascent iterations. Let κ be the proportion of coordinate ascent iterations that do nothing. We can then quantify the quality of a curriculum by $Q(\pi) = \mathbb{E}[\kappa]$ where the expectation is taken with respect to the successive randomly picked coordinate ascent iterations. It is clear from experience that different curricula will have very different qualities $Q(\pi)$.

This reasoning is easily extended to a setup where the teacher chooses each example according to a policy π that takes into account the state of the teacher and the state of the algorithm. We can again define the quality of a policy as $Q(\pi) = \mathbb{E}[\kappa]$ where the expectation is taken on both the successive randomly picked coordinate ascent iterations and the successive training examples selected by the policy. This setup actually describes an *active learning* model similar to that

described in Section 4.3. Hence, Theorems 6 and 7 still apply: LaSVM with active learning also tracks the sequence of SVM solutions. In Section 4.3, we have empirically shown that example selection policies have a considerable impact on the quality of these SVM solutions, and therefore on the performances of LaSVM.

4.5 Summary

This chapter first presented the Huller, a novel online kernel classifier algorithm that converges to the Hard Margin SVM solution. Experiments suggest that it matches the SVM accuracies after a single pass over the training examples thanks to its original Process/Reprocess strategy. Time and memory requirements are then modest in comparison to state-of-the-art SVMs. However the Huller is limited because it cannot handle properly noisy problems.

We have then refined this work and proposed an online algorithm that converges to the Soft-Margin SVM solution. LaSVM reliably reaches competitive accuracies after performing a single pass over the training examples, outspeeding state-of-the-art SVM solvers, especially when data-size grows. We have also shown how active example selection can yield even faster training, higher accuracies and simpler models using only a fraction of the training examples labels. With its online and active learning properties, LaSVM is nowadays the algorithm of choice when one wants to learn a SVM implementing non-linear kernels on a large data sets. For example, it has been successfully employed to train a SVM for handwritten character recognition on more than 8 millions examples on a single CPU [Loosli *et al.*, 2007].

Leveraging a novel duality lemma, we have finally presented tracking guarantees for approximate incremental SVMs that compare with results about batch SVMs and provide generalization guarantees with no extra-computation. This allowed us to give theoretical clues on why algorithms implementing the Process/Reprocess principle (such as the Huller and LaSVM) perform well in a single pass.

5

Large-Scale SVMs for Structured Output Prediction

Contents

In this chapter, we propose LaRank, an online algorithm for the optimization of the dual formulation of support vector methods for structured output spaces [Altun *et al.*, 2003, Tsochantaridis *et al.*, 2005], designed to present good abilities to handle large-scale training databases. We recall that the issue of structured output prediction as well as previous work are extensively presented in Section 2.2.

Following the work on fast optimization of Support Vector Machines of Chapter 4, this novel algorithm performs SMO-like optimization steps over pairs of dual variables, and alternates between unseen patterns and currently support patterns. As a result:

- LaRank generalizes better than perceptron-based algorithms. In fact, LaRank provides the performance of batch algorithms because it solves the same optimization problem.
- LaRank achieves nearly optimal test error rates after a single pass over the randomly reordered training set. Therefore, LaRank offers the practicality of any online algorithm.

LaRank is similar in spirit to LaSVM presented in Section 4.2 since they both implement a PROCESS/REPROCESS strategy to solve a dual SVM QP. However LaRank tackles a more complex problem involving vast output spaces. As we will see in the following, LaRank must sample the potential support vectors on two levels: (1) among the training inputs and (2) for each input, within its realizable outputs. It is intractable to perform these sampling only based on gradient information as LaSVM does. LaRank must treat the support vectors differently.

This chapter follows three steps. First, Section 5.1 introduces the general LaRank algorithm and its theoretical properties. Then, Section 5.2 and Section 5.3 respectively present its application to the benchmarked tasks of multiclass classification and sequence labeling, discussing implementation details as well as experimental results. The work presented in this chapter has been the object of two publications (e.g. [Bordes *et al.*, 2007] and [Bordes *et al.*, 2008]).

5.1 Structured Output Prediction with LaRank

As detailed in Section 2.2, the recovery of the structured output associated to an input pattern p can be carried out using a prediction function such as

$$\begin{array}{rcl} f(p) & = & \arg\max_{c\in\mathcal{C}} S(p,c) \\ & = & \arg\max_{c\in\mathcal{C}} \langle w, \Phi(p,c)\rangle \end{array} \tag{5.1}$$

with $\Phi(p,c)$ mapping the pair (p,c) into a suitable feature space endowed with the dot product $\langle\cdot,\cdot\rangle$. This feature mapping function Φ is usually implicitly defined by a joint kernel function

$$K(p,c,\bar{p},\bar{c}) = \langle \Phi(p,c), \Phi(\bar{p},\bar{c})\rangle \ . \tag{5.2}$$

Given a training set of pattern-output pairs $(p_i,c_i) \in \mathcal{P}\times\mathcal{C}$, $i=1,\ldots,n$, it has been shown that the parameter vector w can be learnt by solving the following Quadratic Programming problem:

$$\begin{array}{c} \max\limits_{\boldsymbol{\beta}} \ -\sum\limits_{i,c} \Delta(c,c_i)\beta_i^c - \frac{1}{2}\sum\limits_{i,j,c,\bar{c}} \beta_i^c\beta_j^{\bar{c}} K(p_i,c,p_j,\bar{c}) \\ \text{subject to} \left\{ \begin{array}{ll} \forall i & \forall c \ \ \beta_i^c \le \delta(c,c_i)\, C \\ \forall i & \sum\limits_c \beta_i^c = 0 \end{array} \right. \end{array} \tag{5.3}$$

where $\Delta(c,c_i)$ is the true loss incurred by predicting c instead of the desired output c_i and $\delta(c,\bar{c})$ is 1 when $c=\bar{c}$ and 0 otherwise. The prediction function is then defined as

$$f(p) = \arg\max_{c\in\mathcal{C}} \sum_{i,\bar{c}} \beta_i^{\bar{c}} K(p_i,\bar{c},p,c).$$

During the execution of the optimization algorithm, we call *support vectors* all pairs (p_i,c) whose associated coefficient β_i^c is non zero; we call *support patterns* all patterns p_i that appear in a support vector.

The LaRank algorithm stores the following data:

- The set $\mathcal{S}$ of the current support vectors.
- The coefficients β_i^c associated with the support vectors $(p_i,c) \in \mathcal{S}$. This encodes the solution since all the other $\boldsymbol{\beta}$ coefficients are zero.

- The derivatives $g_{i,c}$ of the dual objective function with respect to the coefficients β_i^c associated with the support vectors $(p_i, c) \in \mathcal{S}$

$$g_{i,c} = \Delta(c, c_i) - \sum_{j,\bar{c}} \beta_i^{\bar{c}} K(p_j, \bar{c}, p_i, c) \ . \tag{5.4}$$

Note that caching some gradient values (and update them on the course of learning) only saves training time when non-linear input kernels (i.e. polynomial, RBF, ...) are employed. For linear kernels, computing a fresh derivative or updating a stored one has equivalent costs.

LaRank does not store or even compute the remaining coefficients of the gradient. In general, these missing derivatives are not zero because the gradient is not sparse but storing the whole gradient is impracticable when dealing with structured output prediction. As a consequence, for the sake of tractability, we forbid LaRank to use full gradient information to perform its updates.

5.1.1 Elementary Step

Problem (5.3) lends itself to a simple iterative algorithm whose elementary steps are inspired by the well known sequential minimal optimization (SMO) algorithm [Platt, 1999].

Algorithm 17 SmoStep $(i,\ c_+,\ c_-)$

1: Retrieve or compute g_{i,c_+}.
2: Retrieve or compute g_{i,c_-}.
3: Let $\lambda^u = \frac{g_{i,c_+} - g_{i,c_-}}{||\Phi(p_i,c_+) - \Phi(p_i,c_-)||^2}$
4: Let $\lambda = \max\left\{ 0,\ \min(\ \lambda^u,\ C\,\delta(c_+, c_i) - \beta_i^{c_+}\ )\right\}$
5: Update $\beta_i^{c_+} \leftarrow \beta_i^{c_+} + \lambda$ and $\beta_i^{c_-} \leftarrow \beta_i^{c_-} - \lambda$
6: Update $\mathcal{S}$ according to whether $\beta_i^{c_+}$ and $\beta_i^{c_-}$ are zero.
7: Update gradients: $\forall (p_j, c) \in \mathcal{S},\quad g_{j,c} \leftarrow g_{j,c} + \lambda\ (K(p_i, c_+, p_j, c) - K(p_i, c_-, p_j, c))$

Each iteration starts with the selection of one pattern p_i and two outputs c_+ and c_-. The elementary step modifies the coefficients $\beta_i^{c_+}$ and $\beta_i^{c_-}$ by opposite amounts,

$$\begin{array}{rcl} \beta_i^{c_+} & \longleftarrow & \beta_i^{c_+} + \lambda \\ \beta_i^{c_-} & \longleftarrow & \beta_i^{c_-} - \lambda \end{array} \tag{5.5}$$

where $\lambda \geq 0$ maximizes the dual objective function (5.3) along the direction defined by $\Phi(p_i, c_+) - \Phi(p_i, c_-)$ and subject to the constraints. This optimal value is easily computed by first calculating the unconstrained optimum

$$\lambda^u = \frac{g_{i,c_+} - g_{i,c_-}}{||\Phi(p_i, c_+) - \Phi(p_i, c_-)||^2} \tag{5.6}$$

and then enforcing the constraints

$$\lambda = \max\left\{ 0,\ \min(\ \lambda^u,\ C\,\delta(c_+, c_i) - \beta_i^{c_+}\ )\right\} \tag{5.7}$$

Finally, if the input kernel is non-linear, the stored derivatives $g_{j,c}$ are updated to reflect the coefficient update. This is summarized in Algorithm 17.

5.1.2 Step Selection Strategies

Popular SVM solvers based on SMO select successive steps by choosing the pair of coefficients that defines the feasible search direction with the highest gradient (see Section 2.1.2 or 4.2). We cannot use this strategy because we have chosen to store only a small fraction of the gradient.

Stochastic algorithms inspired by the perceptron perform quite well by successively updating coefficients determined by randomly picking training patterns. For instance, in a multiclass context, [Taskar, 2004] (Section 6.1) iterates over the randomly ordered patterns: for each pattern p_i, he computes the scores $S(p_i, c)$ for all outputs and runs SmoStep on the two most violating outputs, that is, the outputs that define the feasible search direction with the highest gradient.

In the context of binary classification, our work on the Huller (Section 4.1) shows that such perceptron-inspired updates lead to a slow optimization of the dual because the coefficients corresponding to the few support vectors are not updated often enough. We suggest instead to alternatively update the coefficient corresponding to a fresh random example and the coefficient corresponding to an example randomly chosen among the current support vectors. The related LaSVM algorithm (Section 4.2) also alternates steps exploiting a fresh random training example and steps exploiting current support vectors selected using the gradient.

We now extend this idea to the structured output formulation. Since this problem has both *support vectors* and *support patterns*, we define three ways to select a triple (i, c_+, c_-) for the elementary SmoStep.

Algorithm 18 ProcessNew (p_i)

1: **if** p_i is a support pattern **then exit.**
2: $c_+ \leftarrow c_i$.
3: $c_- \leftarrow \arg\min_{c \in \mathcal{C}} g_{i,c}$
4: Perform SmoStep $(i,\ c_+,\ c_-)$

Algorithm 19 ProcessOld

1: Randomly pick a *support pattern* p_i.
2: $c_+ \leftarrow \arg\max_{c \in \mathcal{C}} g_{i,c}$ subject to $\beta_i^c < C\,\delta(c, c_i)$
3: $c_- \leftarrow \arg\min_{c \in \mathcal{C}} g_{i,c}$
4: Perform SmoStep $(i,\ c_+,\ c_-)$

Algorithm 20 Optimize

1: Randomly pick a *support pattern* p_i.
2: Let $\mathcal{C}_i = \{\ c \in \mathcal{C}$ such that $(p_i, c) \in \mathcal{S}\ \}$
3: $c_+ \leftarrow \arg\max_{c \in \mathcal{C}_i} g_{i,c}$ subject to $\beta_i^c < C\,\delta(c, c_i)$
4: $c_- \leftarrow \arg\min_{c \in \mathcal{C}_i} g_{i,c}$
5: Perform SmoStep $(i,\ c_+,\ c_-)$

- ProcessNew (Algorithm 18) operates on a pattern p_i that is *not a support pattern*. It chooses the outputs c_+ and c_- that define the feasible direction with the highest gradient. Since all the β_i^c are zero, c_+ is always c_i. Choosing of c_- consists of finding $\arg\max_c S(p_i, c)$ since equation (5.4) holds.
- ProcessOld (Algorithm 19) randomly picks a *support pattern* p_i. It chooses the outputs c_+ and c_- that define the feasible direction with the highest gradient. The determination

of c_+ mostly involves labels c such that $\beta_i^c < 0$, for which the corresponding derivatives $g_{i,c}$ are known. The determination of c_- again consists of computing $\arg\max_c S(p_i, c)$.

- Optimize (Algorithm 20) resembles ProcessOld but picks the outputs c_+ and c_- among those that correspond to existing *support vectors* (p_i, c_+) and (p_i, c_-). Using the gradient is fast because the relevant derivatives are already known and their number is moderate.

Similarly to the Reprocess operation of LaSVM, ProcessOld and Optimize can remove support vectors from the expansion as the SmoStep can nullify β coefficients. The ProcessNew operation is closely related to the perceptron algorithm. It can be interpreted as a stochastic gradient update for the minimization of the generalized margin loss ([Le Cun *et al.*, 2007], Section 2.2.3), with a step size adjusted according to the curvature of the dual [Hildreth, 1957]. [Crammer and Singer, 2003] use a very similar approach for the MIRA algorithm.

5.1.3 Scheduling

Our previous algorithms on binary classification (Huller and LaSVM in Chapter 4) simply alternate two step selection strategies according to a fixed schedule. However some results suggest that the optimal schedule might be in fact data-dependent. We thus propose two kinds of scheduling strategies for the LaRank algorithm.

Fixed Schedule

This is the simplest approach, closely related to the Huller and LaSVM. We call Reprocess the combination of one ProcessOld step followed by ten Optimize steps. The fixed schedule consists in repeatedly performing one ProcessNew step followed by a predefined number n_{R} of Reprocess combinations. The number n_{R} depends on each problem and has to be determined like an hyper-parameter using a validation set. The LaRank algorithm with fixed schedule is presented in Algorithm 21.

Algorithm 21 LaRank with fixed schedule

input: n_{R}.
$\mathcal{S} \leftarrow \emptyset$.
loop
  Randomly reorder the training examples.
  $k \longleftarrow 1$.
  while $k \leq n$ **do**
    Perform ProcessNew (p_k).
    $k \leftarrow k + 1$.
    for $r = 1, \ldots n_{\mathrm{R}}$ **do**
      Perform Reprocess, i.e. 1 ProcessOld + 10 Optimize.
    end for
  end while
end loop
return

Besides its simplicity, this scheduling type is convenient because one controls exactly the number of optimization steps: for one epoch on n fresh examples, at most $n(1 + 11n_{\mathrm{R}})$ SmoStep are performed, i.e. 1 ProcessNew + n_{R} Reprocess (= 1 ProcessOld + 10 Optimize) per example. It is worth noting that this number is linear with the data set size.

Notice that only performing ProcessNew steps (i.e. $n_{\mathrm{R}} = 0$) yields a typical *passive-aggressive* online algorithm [Crammer *et al.*, 2006]. Therefore, the Reprocess operation is the element that lets LaRank match the test accuracy of batch optimization after a single sweep over the training data (see experiments in Sections 5.2 and 5.3).

Adaptive Schedule

The previously defined schedule requires to tune the extra parameter n_{R}. Furthermore, nothing indicates that a strategy fixed during the whole training phase is the best choice: n_{R} might need to be adjusted on the course of learning. Experiments on the influence of Reprocess operations for LaSVM (displayed in Section 4.2.5) even suggest that a rigid schedule might not be optimal.

Algorithm 22 LaRank with adaptive schedule

1: $\mathcal{S} \leftarrow \emptyset,\ \mu.$
2: $r_{\textsc{Optimize}}, r_{\textsc{ProcessOld}}, r_{\textsc{ProcessNew}} \leftarrow 1.$
3: **loop**
4: Randomly reorder the training examples.
5: $k \longleftarrow 1.$
6: **while** $k \leq n$ **do**
7: Pick operation s with odds proportional to r_s.
8: **if** $s =$ Optimize **then**
9: Perform Optimize.
10: **else if** $s =$ ProcessOld **then**
11: Perform ProcessOld.
12: **else**
13: Perform ProcessNew (p_k).
14: $k \leftarrow k + 1.$
15: **end if**
16: $r_s \leftarrow \max\left(0.05\ \frac{\text{dual increase}}{\text{duration}} + 0.95\, r_s,\ \mu\right).$
17: **end while**
18: **end loop**
19: **return**

Actually, one might like to select at each step an operation that causes a large increase of the dual in a small amount of time. We thus propose the following adaptive schedule for LaRank (Algorithm 22). For each operation type, LaRank maintains a running estimate of the average ratio of the dual increase over the duration (line 16). Running times are measured; dual increases are derived from the value of λ computed during the elementary step. The small tolerance μ keeps estimates to reasonable values (usually $\mu = 0.05$). Each iteration of the LaRank algorithm randomly selects which operation to perform with a probability proportional to these estimates.

5.1.4 Stopping

Neither Algorithm 21 nor Algorithm 22 specify a criterion for stopping their outer loops. LaRank is designed to have a full online behavior and excellent results are obtained by performing just one outer loop iteration (epoch). Hence, the default behavior of LaRank is to perform a single epoch, that is to say, a single pass over the randomly ordered training examples.

However LaRank solves the exact convex QP problem (5.3) equivalent to that defined in [Tsochantaridis *et al.*, 2005]. Similarly to LaSVM, it can thus be used in a batch setting by

looping several times over a closed training set. In this case, convenient stopping criteria include exploiting the duality gap ([Schölkopf and Smola, 2002], Section 10.1.1) and monitoring the performance measured on a validation set. We use the name LaRankGap to indicate that we iterate LaRank (Algorithm 21 or 22) until the difference between the primal cost (2.18) and the dual cost (2.21) (defined in Chapter 2) becomes smaller than C. However, computing the duality gap can become quite expensive and involve tremendous increases of training time for LaRankGap on large problems. In such cases, the full online version of LaRank is the best choice.

5.1.5 Theoretical Analysis

This section displays theoretical results concerning the LaRank algorithm: a bound on the number of support vectors and another on the regret. These bounds do not depend on the chosen schedule and are valid for both Algorithms 21 and 22.

Correctness and Complexity

Leveraging the theoretical framework of Appendix B can provide convergence results for LaRank. Let $\rho_{max} = \max_{i,c} ||\Phi(p_i, c) - \Phi(p_i, c_i)||^2$ and let κ, τ, η be small positive tolerances. We assume that the algorithm implementation enforces the following properties:

- SmoStep exits when $g_{i,c_+} - g_{i,c_-} \leq \tau$.
- Optimize and ProcessOld chooses c_+ among the c that satisfy $\beta_i^c \leq C\,\delta(c, c_i) - \kappa$.
- LaRank makes sure that every operation has probability greater than η to be selected at each iteration (trivial for Algorithm 21 and ensured by the μ parameter for Algorithm 22).

We refer to this as the (κ, τ, η)-algorithm.

Theorem 9 *With probability 1, the (κ, τ, η)-algorithm reaches a $\kappa\tau$-approximate solution of problem* (5.8)*, with adding no more than* $\max\{\frac{2\rho_{max} nC}{\tau^2}, \frac{2nC}{\kappa\tau}\}$ *support vectors.*

Proof The convergence is a consequence from Theorem 28 from Appendix B. To apply this theorem, we must prove that the directions defined by (5.5) form a *witness family* for the polytope defined by the constraints of problem (5.3). This is the case because this polytope is a product of n polytopes for which we can apply Proposition 18 from Appendix B. Then, we must ensure that all directions satisfying the first two conditions would be eventually picked. This is guaranteed by the third condition. The number of support vectors is then bounded using a technique similar to that of [Tsochantaridis *et al.*, 2005]. □

The bound on the number of support vectors is also one on the number of *successful* SmoStep required to converge: a successful SmoStep corresponds to a call to Algorithm 17 which actually modifies the pair of β coefficients (i.e. $\lambda \neq 0$). Interestingly, this bound is linear in the number of examples and does not depend on the possibly large number of outputs.

Regret Bound

When learning in a single pass, the LaRank algorithm performs an iterative optimization of the dual, where only the parameters corresponding to already seen examples can be modified at each step. In this section, we extend the primal-dual view of online learning of [Shalev-Shwartz and Singer, 2007a] to structured predictors (i.e. online optimizers of equation (5.3)) to obtain online learning rates.

Regret Bound for Online Structured Predictors The learning rates are expressed with the notion of *regret* defined by the difference between the mean loss incurred by the algorithm on the course of learning and the empirical loss of a given weight vector,

$$\text{regret}(n, w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w_i, (p_i, c_i)) - \frac{1}{n}\sum_{i=1}^{n} \ell(w, (p_i, c_i))$$

with w_i the primal weight vector before seeing the i-th example, and $\ell(w, (p, c))$ the loss incurred by any weight vector w on the example (p, c). In our setup, the loss $\ell(w_i, (p_i, c_i))$ is defined as

$$\ell(w_i, (p_i, c_i)) = \max\left(0,\ \max_{c \in \mathcal{C}} \Delta(c_i, c) - \langle w, \Phi(p_i, c_i) - \Phi(p_i, c)\rangle\right) .$$

The following theorem gives a bound on the regret for any algorithm performing an online optimization of the dual of equation (5.3):

Theorem 10 *Assume that for all i, the dual increase after seeing example (p_i, c_i) is at least $C\mu_\rho(\ell(w_i, (p_i, c_i)))$, with*

$$\mu_\rho(x) = \frac{1}{\rho C}\min(x, \rho C)\left(x - \frac{1}{2}\min(x, \rho C)\right)$$

then, we have:

$$\forall w,\ \textit{regret}(n, w) \leq \frac{||w||^2}{2nC} + \frac{\rho C}{2}.$$

Proof The proof exactly follows Section 5 of [Shalev-Shwartz and Singer, 2007a]. Let's denote $P_t(w)$ and $D_t(w)$ the primal and dual after seeing t examples for any weight vector w. The function μ_ρ is invertible on $\mathbb{R}_+$ and its inverse is

$$\mu_\rho^{-1}(x) \begin{cases} x + \frac{\rho C}{2} & \text{if } x \geq \frac{\rho C}{2} \\ \sqrt{2\rho C x} & \text{otherwise .} \end{cases}$$

As $D_{t+1}(w_{t+1}) - D_t(w_t) \geq C\mu_\rho(\ell(w_t, (p_t, c_t)))$ and assuming $D_0(w_0) = 0$, we deduce

$$D_{n+1}(w_{n+1}) \geq C\sum_{t=1}^{n} \mu_\rho(\ell(w_t, (p_t, c_t))) .$$

By the weak duality theorem, $\forall w\ ,\ P_{n+1}(w) \geq D_{n+1}(w)$, and

$$\forall w\ \ \frac{||w||^2}{2C} + \sum_{t=1}^{n} \ell(w, (p_t, c_t) \geq \sum_{t=1}^{n} \mu_\rho(\ell(w_t, (p_t, c_t))) .$$

As μ_ρ is a convex function,

$$\forall w\ \ \frac{||w||^2}{2C} + \sum_{t=1}^{n} \ell(w, (p_t, c_t) \geq \mu_\rho\left(\sum_{t=1}^{n} \ell(w_t, (p_t, c_t))\right) .$$

Both sides of the above inequality are non-negative, μ_ρ is invertible, μ_ρ^{-1} is monotonically increasing, then

$$\forall w\ \ \mu_\rho^{-1}\left(\frac{||w||^2}{2C} + \sum_{t=1}^{n} \ell(w, (p_t, c_t)\right) \geq \sum_{t=1}^{n} \ell(w_t, (p_t, c_t)) .$$

Since $\forall x,\ \mu_\rho^{-1}(x) \leq x + \frac{\rho C}{2}$,

$$\forall w\ \ \frac{||w||^2}{2nC} + \frac{\rho C}{2n} \geq \frac{1}{n}\sum_{t=1}^{n} \ell(w_t, (p_t, c_t)) - \frac{1}{n}\sum_{t=1}^{n} (\ell(w, (p_t, c_t)) .$$

□

The crucial point of this theorem is to directly relate the dual increase when seeing an example and the regret bound: the more we can prove that the dual increases on the course of learning, the more we can have guarantees on the performance.

Application to LaRank The following result allows to use Theorem 10 to bound the regret for the LaRank algorithm:

Proposition 11 *For a given i, the dual increase after performing a* ProcessNew *step on example (p_i, c_i) is equal to*

$$C\mu_{\rho^i}(\ell(w_i, (p_i, c_i)))\ ,$$

with $\rho^i = ||\Phi(p_i, c_i) - \Phi(p_i, c_i^*)||^2$ *and* $c_i^* = \arg\max_{c \in \mathcal{C}} \left(\Delta(c_i, c) + \langle w_i, \Phi(p_i, c)\rangle\right)$.

Proof $D_t(w)$ still denotes the dual after seeing t examples. The direct calculation of the dual increase after a ProcessNew step on example (p_t, c_t) yields $D_{t+1}(w_{t+1}) - D_t(w_t) = \lambda\ell(w_t(p_t, c_t)) - \rho_t(\lambda)^2/2$ with $\lambda = \min(C, \ell(w_t, (p_t, c_t))/\rho_t)$ and $\rho_t = ||\Phi(p_t, c_t) - \Phi(p_t, f(w_t, c_t))||^2$. Using the definition of μ_ρ, $D_{t+1}(w_{t+1}) - D_t(w_t) = C\mu_{\rho_t}(\ell(w_t(p_t, c_t)))$.

Since neither ProcessOld nor Optimize can decrease the dual, the whole LaRank algorithm increases the dual by at least $C\mu_{\rho^i}(\ell(w_i, (p_i, c_i)))$ after seeing example i. Moreover, as μ_ρ monotonically decreases with ρ theorem 10 can be applied to LaRank with $\rho = \max_i \rho^i$. □

Interpretation Proposition 11 first shows that the first epoch of LaRank has the same guarantees (in terms of regret) than a typical *passive-aggressive* algorithm as the latter is equivalent to performing only ProcessNew operations.

In addition, Theorem 10 provides a partial justification of the ProcessOld and Optimize functions. Indeed, it expresses that we can relate the dual increase to the regret. As such, if, for instance, ProcessOld and Optimize operations bring a dual increase of the same order of magnitude as ProcessNew operations at each round, then the regret of LaRank would be typically two times smaller than the current bound. Although we do not have any analytical results concerning the dual increase ratio between ProcessNew and ProcessOld/Optimize operations, the theorem suggests that the true regret of LaRank should be much smaller than the bound. We can also note that the tracking guarantees established in Section 4.4 for LaSVM could be translated to LaRank.

The bound is also informative to compare online to batch learning. Indeed, if we consider the examples (p_i, c_i) in the regret bound to be the training set, Theorem 10 relates the online error with the error of the batch optimal. Then, we can claim that the online error of LaRank will not be too far from the batch optimal trained with the same set of examples.

We have introduced LaRank, an online algorithm for structured output prediction inspired by LaSVM, and we have exhibited its nice theoretical properties. The following sections display how it can be applied to two concrete cases: multiclass classification (Section 5.2) and sequence labeling (Section 5.3).

5.2 Multiclass Classification

As explained in Section 2.2.1, the formalism of SVMs for structured outputs derives from a model originally destined to multiclass classification. Presenting a good behavior (especially, reduced computational costs and memory needs) on multiclass problems is therefore a key condition for any large-scale structured output prediction candidate. This is the reason why we experience the behavior of LaRank on this task at first.

5.2.1 Multiclass Factorization

For the problem of multiclass classification, a pattern p is simply a vector similar to those $x \in \mathcal{X}$ of the binary classification case and the output outputs c correspond to atomic class labels $y \in \mathcal{Y}$, where $\mathcal{Y}$ can contain more than two elements. The joint kernel function (5.2) is simply $K(p,c,\bar{p},\bar{c}) = k(x,\bar{x})\,\delta(y,\bar{y})$, where $k(x,\bar{x})$ is a kernel defined on the inputs, and where $\delta(y,\bar{y})$ is 1 if $y = \bar{y}$ and 0 otherwise.

The dual problem (5.3) can be drastically simplified and becomes

$$\max_{\boldsymbol{\beta}} \sum_i \beta_i^{y_i} - \frac{1}{2}\sum_{i,j}\sum_y \beta_i^y \beta_j^y\, k(x_i,x_j) \quad \text{subject to} \begin{cases} \forall i \;\; \forall y \;\; \beta_i^y \le C\delta(y,y_i) \\ \forall i \;\; \sum_y \beta_i^y = 0 \end{cases} \tag{5.8}$$

When there are only two outputs, one can show that this reduces to the standard SVM solution (without bias) presented in Chapter 2.

The prediction function is defined as $f(x) = \arg\max_{y\in\mathcal{Y}} \sum_i \beta_i^y k(x_i,x)$. In standard multiclass problems, the number of classes $|\mathcal{Y}|$ is reasonably small (see Table 5.1 for rough estimates). Solving the arg max is then simply an exhaustive search over all $\mathcal{Y}$.

5.2.2 LaRank Implementation for Multiclass Classification

For multiclass classification, LaRank uses the adaptive schedule (Algorithm 22) as it allows to automatically balance the use of each elementary operation. In order to facilitate timing, we treat sequences of ten Optimize as a single atomic operation.

On most of multiclass classification benchmarks, the use of non-linear input kernels $k(x,\bar{x})$ is required to reach competitive accuracies. Non-linear kernels involves higher complexities. Special implementation care must then be taken for LaRank to remain efficient and so, LaRank caches some useful kernel values. A naive implementation could simply pre-compute all the kernel values $k(x_i,x_j)$. This would be a waste of processing time and memory because the location of the optimum depends only on the fraction of the kernel matrix that involves support patterns. Our code computes kernel values on demand and caches them in sets of the form

$$E(y,i) = \{\, k(x_i,x_j) \;\text{ such that }\; (x_j,y) \in \mathcal{S} \,\}.$$

Although this cache stores several copies of the same kernel values, caching individual kernel values has a higher overhead caused by the extra-costs to retrieve values one by one.

A C++ implementation of LaRank for multiclass classification, featuring the kernel cache and the adaptive schedule, is freely available on the `mloss.org` website under the `GNU Public License` (go to `http://mloss.org/software/view/127/`).

5.2.3 Experiments

This section reports experiments carried out on various multiclass pattern recognition problems in order to well characterize the algorithm behavior. Most methods compared in this section are detailed in Section 2.2.

	Train Ex.	Test Ex.	Classes	Features	C	$k(x, \bar{x})$
Letter	16000	4000	26	16	10	$e^{-0.025\|x-\bar{x}\|^2}$
USPS	7291	2007	10	256	10	$e^{-0.025\|x-\bar{x}\|^2}$
MNIST	60000	10000	10	780	1000	$e^{-0.005\|x-\bar{x}\|^2}$
INEX	6053	6054	18	167295	100	$x \cdot \bar{x}$

Table 5.1: **Data sets and parameters used for the multiclass experiments.**

		Letter	USPS	MNIST	INEX
MCSVM	Test error (%)	2.42	**4.24**	1.44	**26.26**
(stores the full gradient)	Dual	**5548**	**537**	3718	235204
	Training time (sec.)	1200	**60**	**25000**	520
	Kernels ($\times 10^6$)	241	51.2	6908	32.9
SVMstruct	Test error (%)	**2.40**	4.38	**1.40**	**26.25**
(stores partial gradient)	Dual	5495	528	**3730**	**235631**
	Training time (sec.)	23000	6300	265000	14500
	Kernels ($\times 10^6$)	2083	1063.3	158076	n/a[†]
LaRankGap	Test error (%)	**2.40**	4.38	1.44	**26.25**
(stores partial gradient)	Dual	5462	518	3718	235183
	Training time (sec.)	2900	175	82000	1050
	Kernels ($\times 10^6$)	156	13.7	4769	19.3
LaRank	Test error (%)	2.80	**4.25**	**1.41**	27.20
(online)	Dual	5226	503	3608	214224
	Training time (sec.)	**940**	85	30000	**300**
	Kernels ($\times 10^6$)	**55**	**9.4**	**399**	**17.2**

[†] Not applicable because SVMstruct bypasses the cache when using linear kernels.

Table 5.2: **Compared test error rates and training times on multiclass data sets.**

Experimental Setup

Experiments were performed on four data sets briefly described in Table 5.1: Letter and USPS available from the UCI repository,[1] MNIST[2] that we already used in Chapter 4 and INEX, a data set containing scientific articles from 18 journals and proceedings of the IEEE. We use a flat TF/IDF feature space for INEX (see [Denoyer and Gallinari, 2006] for further details).

Table 5.1 also lists our choices for the parameter C and for the kernels $k(x, \bar{x})$. These choices were made on the basis of past experience. We use the same parameters for all algorithms because we mostly compare algorithms that optimize the same criterion. The kernel cache size was 500MB for all experiments.

Comparing Batch Optimizers

Table 5.2 (top half) compares three optimization algorithms for the same dual cost (5.8).

[1] `http://www.ics.uci.edu/~mlearn/databases`.
[2] `http://yann.lecun.com/exdb/mnist`.

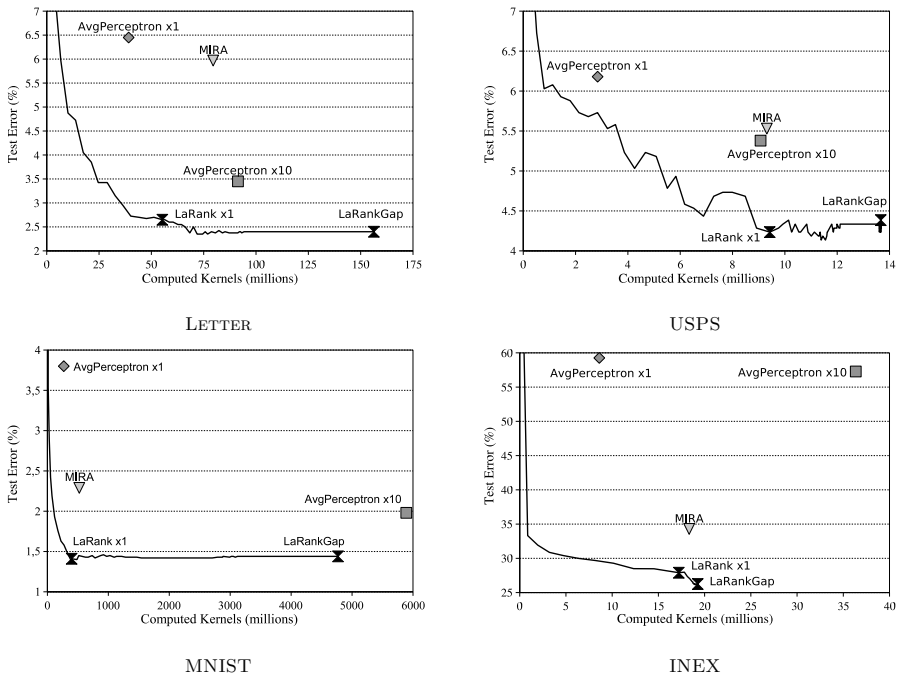


Figure 5.1: **Test error as a function of the number of kernel calculations.** LaRank almost achieves its final accuracy after a single epoch on all data sets..

- MCSVM [Crammer and Singer, 2001] uses the full gradient and therefore cannot be easily extended to handle structured output problems. We have used the MCSVM implementation distributed by the authors.
- SVMstruct [Tsochantaridis *et al.*, 2005] targets structured output problems and therefore uses only a small fraction of the gradient. We have used the implementation distributed by the authors. The authors warn that this implementation has not been thoroughly optimized.
- LaRankGap iterates Algorithm 22 until the duality gap becomes smaller than parameter C. This algorithm only stores a small fraction of the gradient, comparable to that used by SVMstruct.

Both SVMstruct and LaRankGap use small subsets of the gradient coefficients. Although these subsets have similar size, LaRankGap avoids the training time penalty experienced by SVMstruct.

Both SVMstruct and LaRank make heavy use of kernel values involving two support patterns. In contrast, MCSVM updates the complete gradient vector after each step and therefore uses the kernel matrix rows corresponding to support patterns. On our relatively small problems, this stronger memory requirement is more than compensated by the lower overhead of MCSVM's simpler cache structure. However as MCSVM needs to store the whole gradient it cannot scale to structured output prediction where the number of classes is very large.

Comparing Online Learning Algorithms

Table 5.2 (bottom half) also reports the results obtained with a single LaRank epoch. This single pass over the training examples is sufficient to nearly reach the optimal performance. This result is understandable because (*i*) online perceptrons offer strong theoretical guarantees after a single pass over the training examples, and (*ii*) LaRank drives the optimization process by replicating the randomization that happens in the perceptron. This is also coherent with the regret bound presented in Section 5.1.5 and with the performances of LaSVM displayed in Chapter 4.

For each data set, Figure 5.1 shows the evolution of the test error with respect to the number of kernel calculations. The point marked LaRank×1 corresponds to running a single LaRank epoch. The point marked LaRankGap still corresponds to using the duality gap stopping criterion. Figure 5.1 also reports results obtained with two popular online algorithms:

- The points marked AvgPerc×1 and AvgPerc×10 respectively correspond to performing one and ten epochs of the average perceptron algorithm [Freund and Schapire, 1998, Collins, 2002]. Multiple epochs of the averaged perceptron are very effective when the necessary kernel values fit in the cache (first row). Training time increases considerably when this is not the case (second row.)

- The point marked MIRA corresponds to the multiclass *passive-aggressive* algorithm proposed by [Crammer and Singer, 2003]. We have used the implementation provided by the authors as part of the MCSVM package. This algorithm computes more kernel values than AvgPerc×1 because its solution contains more support patterns. Its performance seems sensitive to the choice of kernel: [Crammer and Singer, 2003] report substantially better results using the same code but different kernels.

These results indicate that performing single LaRank epoch is an attractive online learning algorithm. Although LaRank usually runs slower than AvgPerc×1 or MIRA, it provides better and more predictable generalization performance.

Comparing Optimization Strategies

Figure 5.2 shows the error rates and kernel calculations achieved when one restricts the set of operations chosen by Algorithm 22. These results were obtained after a single pass on USPS.

As expected, using only the ProcessNew operation performs like MIRA. The average perceptron requires significantly less kernel calculations because its solution is much more sparse. However, it looses this initial sparsity when one performs several epochs (see Figure 5.1.) Enabling ProcessOld and Optimize significantly reduces the test error. The best test error is achieved when all operations are enabled. The number of kernel calculations is also reduced because ProcessOld and Optimize often eliminate support patterns.

Comparing ArgMax Calculations

The previous experiments measure the computational cost using training time and number of kernel calculations. Most structured output problems require the use of costly algorithms to perform the inference step (e.g. sequence labeling, see Section 5.3). The cost of this arg max calculation is partly related to the required number of new kernel values.

The average perceptron (and MIRA) performs one such arg max calculation for each example it processes. In contrast, LaRank performs one arg max calculation when processing a new example with ProcessNew, and also when running ProcessOld.

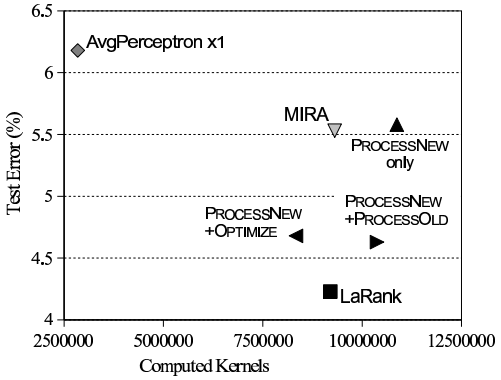


Figure 5.2: **Impact of the** LaRank **operations** (USPS data set).

	Letter	USPS	MNIST	INEX
AvgPerc×1	16	7	60	6
AvgPerc×10	160	73	600	60
LaRank	190	25	200	28
LaRankGap	550	86	2020	73
SVMstruct	141	56	559	78

Table 5.3: **Numbers of** arg max (in thousands).

Table 5.3 compares the number of arg max calculations for various algorithms and data sets.[3] The SVMstruct optimizer performs very well with this metric. AvgPerc and LaRank are very competitive on a single epoch and become more costly when performing many epochs. One epoch is sufficient to reach good performance with LaRank. This is not the case for AvgPerc.

5.3 Sequence Labeling

This section exhibits the specification of LaRank for sequence labeling. This task consists in predicting a sequence of *labels* $(y^1 \dots y^T)$ given an observed sequence of *tokens* $(x^1 \dots x^T)$. This task is a typical example of a structured output learning system. It is a major machine learning task which appears in practical problems in computational linguistics or signal processing.

SVMs for structured outputs can deal with different sorts of structure. However, for sequence labeling, some powerful specific models also exist. For many years, standard methods have been Hidden Markov Models (HMMs) [Rabiner and Juang, 1986], generative systems modelling a sequential task as a Markov process with unobserved states. Conditional Random Fields (CRFs) [Lafferty *et al.*, 2001] are now the state-of-the-art. A CRF is probabilistic framework for labeling and segmenting sequential data. It forms an undirected graphical model that defines a single log-linear distribution over label sequences given a particular observation sequence. Contrary to generative HMMs, CRFs have a conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference. Additionally, CRFs avoid the label bias problem. Hence, they have been shown to outperform HMMs on many sequence labeling tasks. They can be trained either with batch or online methods and thus can scale on large data sets. We use CRFs as reference, in Section 5.3.4.

This section displays the application of LaRank to the task of sequence labeling using two inference schemes detailed in Section 5.3.1. We cast them into the general structured output learning problem in Section 5.3.2 and exhibit the LaRank corresponding derivations in Section 5.3.3. Section 5.3.4 finally displays an empirical evaluation on standard benchmarks for sequence labeling comparing LaRank with CRFs, batch SVM solvers and perceptrons, among others.

[3]The Letter results in Table 5.3 are outliers because the Letter kernel runs as fast as the kernel cache. Since LaRank depends on timings, it often runs ProcessOld when a simple Optimize would have be sufficient.

Even if LaRank can be used in either online or batch mode (see Section 5.1.4), we focus in the remaining of this chapter, on the online version. Indeed, this is clearly the most engaging feature, the one which could lead to a huge leap forward in scalability on large-scale problems.

5.3.1 Representation and Inference

In this section, we use bold characters for sequences such as the sequence of tokens $\mathbf{x} = (x^1 \dots x^T)$ or the sequence of labels $\mathbf{y} = (y^1 \dots y^T)$. Subsequences are denoted using superscripts, as in $\mathbf{y}^{\{t-k..t-1\}} = (y^{t-k} \dots y^{t-1})$. We call $\mathcal{X}$ the set of possible tokens and $\mathcal{Y}$ the set of possible labels, augmented with a special symbol to represent the absence of a label. By convention, a label y^s is the special symbol whenever $s \leq 0$.

Two informal assumptions are crucial for sequence labeling. The first states that a label y^t depends only on the surrounding labels and tokens. The second states that this dependency is invariant with t. These assumptions are expressed through the parametric formulation of the models, and, in the case of probabilistic models, through conditional independence assumptions (e.g. HMMs). Part of the model specification is then the *inference procedure* that recovers the predicted labels for any input sequence. *Exact inference* can be carried out with the Viterbi algorithm. The more efficient *greedy inference*, which predicts the labels in the order of the sequence using the past predictions, can also be competitive in terms of accuracy by considering higher order Markov assumptions.

Thus, an inference procedure assigns a label y^t to each corresponding x^t taking into account the correlations between labels at different positions in the sequence. This work takes into account correlations between $k+1$ successive labels (Markov assumption of order k). More specifically, we assume that the inference procedure determines the predicted label sequence $\mathbf{y}$ on the sole basis of the scores

$$s^t(w, \mathbf{x}, \mathbf{y}) = \left\langle w, \Phi_{\mathrm{g}}\left(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t\right)\right\rangle \quad t = 1...T,$$

where $w \in \mathbb{R}^D$ is a parameter vector and $\Phi_{\mathrm{g}} : \mathcal{X} \times \mathcal{Y}^k \times \mathcal{Y} \to \mathbb{R}^D$ determines the feature space.

Exact Inference

Exact inference maximizes the sum $\sum_{t=1}^{T} s^t(w, \mathbf{x}, \mathbf{y})$ over all possible label sequences $\mathbf{y}$. In this case, for a given input sequence $\mathbf{x}$, the prediction function $\mathbf{f}_{\mathrm{e}}(w, \mathbf{x})$ is then defined by

$$\begin{aligned} \mathbf{f}_{\mathrm{e}}(w, \mathbf{x}) &= \arg\max_{\mathbf{y} \in \mathcal{Y}^T} \sum_{t=1}^{T} s^t(w, \mathbf{x}, \mathbf{y}) \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}^T} \langle w, \Phi_{\mathrm{e}}(\mathbf{x}, \mathbf{y}) \rangle \ , \end{aligned} \tag{5.9}$$

where $\Phi_{\mathrm{e}}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T} \Phi_{\mathrm{g}}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)$.

Greedy Inference

Following [Maes *et al.*, 2007], greedy inference predicts the successive labels y^t in sequence by maximizing $\left\langle w, \Phi_{\mathrm{g}}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)\right\rangle$ where the previously predicted labels $\mathbf{y}^{\{t-k..t-1\}}$ are frozen. For a given input $\mathbf{x}$, the prediction function $\mathbf{f}_{\mathrm{g}}(w, \mathbf{x})$ is then defined by the recursion

$$f_{\mathrm{g}}^t(w, \mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \left\langle w, \Phi_{\mathrm{g}}\big(x^t, \mathbf{f}_{\mathrm{g}}^{\{t-k..t-1\}}(w, \mathbf{x}), y\big)\right\rangle \quad t = 1...T. \tag{5.10}$$

Comparison

Although greedy inference is an approximation of exact inference, their different computational complexity leads to a more nuanced picture. Exact inference solves (5.9) using the Viterbi algorithm. It requires a time proportional to $DT|\mathcal{Y}|^{k+1}$ and becomes intractable when the order k of the Markov assumption increases. On the other hand, the recursion (5.10) runs in time proportional to $DT|\mathcal{Y}|$. Therefore greedy inference is practicable with large k.

In practice, greedy inference with large k can sometimes achieve a higher accuracy than exact inference with Markov assumptions of lower order.

5.3.2 Training

In this section we write the convex optimization problem used for determining the parameter vector for both cases of exact and greedy inference by showing how the general dual problem (5.3) applies to both problems.

Training for Exact Inference

Since the exact inference prediction function (5.9) can be written as $\arg\max_c \langle w, \Phi(p,c)\rangle$, the general formulation (5.3) applies directly. The patterns p_i are the token sequences $\mathbf{x}_i$ and the classes c are complete label sequences $\mathbf{y}$. The feature function $\Phi(p_i, c) = \Phi_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y})$ has been defined in (5.9) and the loss $\Delta(\mathbf{y}, \bar{\mathbf{y}})$ is the Hamming distance between the sequences $\mathbf{y}$ and $\bar{\mathbf{y}}$.

The dual problem is then

$$\begin{aligned} \max_{\boldsymbol{\beta}} \; & -\sum_{i,\mathbf{y}} \Delta(\mathbf{y}, \mathbf{y}_i)\beta_i^{\mathbf{y}} - \frac{1}{2}\sum_{ij}\sum_{\mathbf{y}\bar{\mathbf{y}}} \beta_i^{\mathbf{y}}\beta_j^{\bar{\mathbf{y}}} K_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y}, \mathbf{x}_j, \bar{\mathbf{y}}) \\ & \text{subject to } \begin{cases} \forall i \;\; \forall \mathbf{y} & \beta_i^{\mathbf{y}} \le \delta(\mathbf{y}, \mathbf{y}_i)\, C \\ \forall i & \sum_{\mathbf{y}} \beta_i^{\mathbf{y}} = 0\,. \end{cases} \end{aligned} \tag{5.11}$$

with the kernel matrix $K_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y}, \mathbf{x}_j, \bar{\mathbf{y}}) = \langle \Phi_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y}), \Phi_{\mathrm{e}}(\mathbf{x}_j, \bar{\mathbf{y}})\rangle$.

The solution is then $w = \sum_{i\mathbf{y}} \beta_i^{\mathbf{y}}\, \Phi_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y})$.

Training for Greedy Inference

The greedy inference prediction function (5.10) does not readily have the form $\arg\max_c \langle w, \Phi(p,c)\rangle$ because of its recursive structure. However, each prediction f_{g}^t has the desired form with one pattern p_{it} for each training token x_i^t, and with classes c taken from the set of labels and compared with $\Delta(y, \bar{y}) = 1 - \delta(y, \bar{y})$.

This approach leads to difficulties because the feature function $\Phi(p_{it}, y) = \Phi_{\mathrm{g}}(x_i^t, \mathbf{f}_{\mathrm{g}}^{\{t-k..t-1\}}, y)$ depends on the prediction function. We avoid this difficulty by approximating the predicted labels $\mathbf{f}_{\mathrm{g}}^{\{t-k..t-1\}}$ with the true labels $\mathbf{y}_i^{\{t-k..t-1\}}$.

The corresponding dual problem is then

$$\begin{aligned} \max_{\boldsymbol{\beta}} \; & -\sum_{ity} \Delta(y, y_i^t)\beta_{it}^y - \frac{1}{2}\sum_{itjr}\sum_{y\bar{y}} \beta_{it}^y\beta_{jr}^{\bar{y}} K_{\mathrm{g}}(x_i^t, y, x_j^r, \bar{y}) \\ & \text{subject to } \begin{cases} \forall i,t \;\; \forall y & \beta_{it}^y \le \delta(y, y_i^t)\, C \\ \forall i,t & \sum_y \beta_{it}^y = 0\,. \end{cases} \end{aligned} \tag{5.12}$$

with the kernel matrix $K_{\mathrm{g}}(x_i^t, y, x_j^r, \bar{y}) = \left\langle \Phi_{\mathrm{g}}(x_i^t, \mathbf{y}_i^{\{t-k..t-1\}}, y)\,,\; \Phi_{\mathrm{g}}(x_j^r, \mathbf{y}_j^{\{r-k..r-1\}}, \bar{y}) \right\rangle$.

The solution is then $w = \sum_{ity} \beta_{it}^y\, \Phi_{\mathrm{g}}(x_i^t, \mathbf{y}_i^{\{t-k..t-1\}}, y)$.

Discussion

Both dual problems (5.11) and (5.12) are defined using very different sets of coefficients $\boldsymbol{\beta}$. Experiments (Section 5.3.4) show considerable differences in sparsity. Yet the two kernel matrices K_{e} and K_{g} generate parameter vectors w in the same feature space which is determined by the choice of the feature function Φ_{g}, or equivalently the choice of the kernel K_{g}.

We use the following kernels in the rest of this paper.

$$K_{\mathrm{g}}(x_i^t, y, x_j^r, \bar{y}) = \delta(y, \bar{y})\Big(k(x_i^t, x_j^r) + \sum_{s=1}^{k} \delta(y_i^{t-s}, \bar{y}_j^{r-s})\Big)\,,$$

$$K_{\mathrm{e}}(\mathbf{x}_i, \mathbf{y}, \mathbf{x}_j, \bar{\mathbf{y}}) = \sum_{tr} \delta(y^t, \bar{y}^r)\Big(k(x_i^t, x_j^r) + \sum_{s=1}^{k} \delta(y^{t-s}, \bar{y}^{r-s})\Big)\,,$$

where $k(x, \bar{x}) = \langle x, \bar{x}\rangle$ is a linear kernel defined on the tokens. These two kernels satisfy the identity $\Phi_{\mathrm{e}}(\mathbf{x}, \mathbf{y}) = \sum_i \Phi_{\mathrm{g}}(x^t, \mathbf{y}^{\{t-k..t-1\}}, y^t)$ by construction. Furthermore, the exact inference kernel K_{e} is precisely the kernel proposed in [Altun *et al.*, 2003].

The greedy kernel approximates the predicted labels with the true labels. The same approximation was used in LaSO [Daumé III and Marcu, 2005] and in the first iteration of SEARN [Daumé III *et al.*, 2005]. In the context of an online algorithm, other approximations would have been possible, such as the sequence of predicted labels for the previous values of the parameter. However, the simpler approximation works slightly better in our experiments.

5.3.3 LaRank Implementations for Sequence Labeling

We denote LaRankExact, the LaRank algorithm adapted for solving the dual problem (5.11) for exact inference, and LaRankGreedy the one for solving the dual problem (5.12) for greedy inference. These algorithms stop after a single epoch over the training set. The suffix Gap is added when an algorithm loops several times until the duality gap is smaller than C.

The LaRank algorithm using an adaptive schedule (Algorithm 22) works well for simple multiclass problems. However, we had mixed experiences with the exact inference models, because the ProcessOld operations incur a penalization in terms of computation time due to the Viterbi algorithm. In the end, ProcessOld was not sufficiently applied, leading to poor performance. For this reason, we chose to use a fixed schedule (Algorithm 21) for both LaRankGreedy and LaRankExact. A linear kernel is used for the inner products between tokens. Consequently, no kernel cache is required for either LaRankExact or LaRankGreedy. Storing the gradients is also useless as, in this case, the computational cost of a gradient update and a fresh computation are equivalent.

C++ implementations of both LaRankExact and LaRankGreedy are freely available on the `mloss.org` website under the `GNU Public License`:

- LaRankExact: `http://mloss.org/software/view/198/`
- LaRankGreedy: `http://mloss.org/software/view/199/`

The regret we consider in Section 5.1.5 does not match the true applicative setting of greedy inference. Indeed, we consider in the regret bound a set of examples that is fixed independently of the parameter vector w with which we compare. But on test examples the greedy inference scheme uses the past predictions instead of the true labels. Nevertheless the partial justification for the Reprocess (ProcessOld +Optimize) function is still valid.

Finally, we can remark that the combination of a fixed schedule, a linear kernel and no storage of gradient values to update allows the amount of computations to be performed by LaRank at

each iteration to be identical on the course of learning. Hence, both algorithms enjoy a linear scaling of training time w.r.t the training set size. This asymptotical guarantee, a key aspect for a large-scale algorithm, is observed in practice (see next section).

5.3.4 Experiments

This section reports experiments performed on various label sequence learning tasks to study the behavior of LaRank. Since such tasks are common in the recent literature, we focus on fully supervised tasks where labels are provided for every time index. After presenting the experimental tasks we chose, we compare the performances of LaRankExact and LaRankGreedy to both batch and online methods to empirically validate their efficiency. We then investigate how the choice of the inference method influences the performances.

Experimental Setup

Experiments were carried out on three data sets. The Optical Character Recognition data set (OCR) contains handwritten words, with average length of 8 characters, written by 150 human subjects and collected by [Kassel, 1995]. This is a small data set for which the performance evaluation is performed using 10-fold cross-validation. The CHUNKING data set from the CoNLL 2000 shared task[4] consists of sentences divided in syntactically correlated segments or chunks. This data set has more than 75,000 input features. The Wall Street Journal data set[5] (WSJ) is a larger data set with around 1 million words in more than 40,000 sentences and with a large number of features. The labels associated with each word are "part-of-speech" tags.

Table 5.4 summarizes the main characteristics of these three data sets and specifies the parameters we have used for both batch and online algorithms: the constant C, the number n_{R} of REPROCESS steps for each PROCESSNEW step, and the order k of the Markov assumptions. They have been chosen by cross-validation for the batch setting, online algorithms using the same parameters as their batch counterparts. Exact inference algorithms such as LaRankExact are limited to first order Markov assumptions for tractability reasons.

General Performances

We report the training times for a number of algorithms as well as the percentage of correctly predicted labels on the test sets (for CHUNKING, we also provide F1 scores on test sets). Results for exact inference algorithms are reported in Table 5.5. Results for greedy inference algorithms are reported in Table 5.6. Some discussed methods are detailed in Section 2.2.

Batch Counterparts Our main points of comparison are the prediction accuracies achieved by batch algorithms that fully optimize the same dual problems as our online algorithms. In the case of exact inference, our baseline is given by the SVMstruct results using the cutting plane optimization algorithm [Tsochantaridis *et al.*, 2005] (described in Section 2.2). In the case of greedy inference, the batch baseline is simply LaRankGreedyGap.

Tables 5.5 and 5.6 show that both LaRankGreedy and LaRankExact reach competitive testing set performances relative to these baselines while saving a lot of training time.

Figure 5.3 depicts relative time increments. Denoting t_0 the running time of a model on a small portion of the training set of size s_0, the time increment on a training set of size s is defined as t_s/t_0. We also define the corresponding size increment as s/s_0. This allows to represent scaling

[4] http://www.cnts.ua.ac.be/conll2000/chunking/
[5] http://www.cis.upenn.edu/~treebank/

	Training Set Sequences(Tokens)	Testing Set Sequences(Tokens)	Classes	Features	C	LaRankGreedy n_R	LaRankGreedy k	LaRankExact n_R	LaRankExact k
OCR	650 (∼4,600)	5500 (∼43,000)	26	128	0.1	5	10	10	1
Chunking	8,931 (∼212,000)	2,012 (∼47,000)	21	∼76,000	0.1	1	2	5	1
WSJ	42,466 (∼1,000,000)	2,155 (∼53,000)	44	∼130,000	0.1	1	2	5	1

Table 5.4: **Data sets and parameters used for the sequence labeling experiments.**

		OCR	Chunking	*(F1 score)*	WSJ
CRF	Test. accuracy (%)	-	96.03	*(93.75)*	96.75
(batch)	Train. time (sec.)	-	568		3,400
SVMstruct	Test. accuracy (%)	78.20	95.98	*(93.64)*	96.81
(batch)	Train. time (sec.)	180	48,000		350,000
CRF	Test. accuracy (%)	-	95.26	*(92.47)*	94.42
(online)	Train. time (sec.)	-	30		240
PerceptronExact	Test. accuracy (%)	51.44	93.74	*(89.31)*	91.49
(online)	Train. time (sec.)	0.2	10		180
PAExact	Test. accuracy (%)	56.13	95.15	*(92.21)*	94.67
(online)	Train. time (sec.)	0.5	15		185
LaRankExact	Test. accuracy (%)	75.77	95.82	*(93.34)*	96.65
(online)	Train. time (sec.)	4	130		1380

Table 5.5: **Compared accuracies and times of methods using exact inference.**

		OCR	Chunking	*(F1 score)*	WSJ
LaRankGreedyGap	Test. accuracy (%)	83.77	95.86	*(93.59)*	96.63
(batch)	Train. time (sec.)	15	490		9,000
PerceptronGreedy	Test. accuracy (%)	51.82	93.24	*(88.84)*	92.70
(online)	Train. time (sec.)	0.05	3		10
PAGreedy	Test. accuracy (%)	61.23	94.61	*(91.55)*	94.15
(online)	Train. time (sec.)	0.1	5		25
LaRankGreedy	Test. accuracy (%)	81.15	95.81	*(93.46)*	96.46
(online)	Train. time (sec.)	1.4	20		175

Table 5.6: **Compared accuracies and times of methods using greedy inference.**

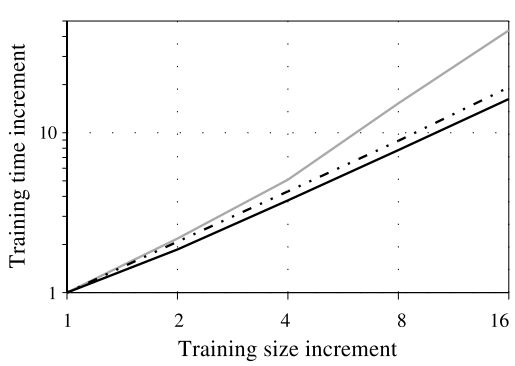


Figure 5.3: **Scaling in time on** CHUNKING **data set.** (log-log plot) Solid black line: LaRankGreedy, Dashed black line: LaRankExact, Gray line: SVMstruct.

	CHUNKING	WSJ
SVMstruct (batch)	1360	9072
PAExact (online)	443	2122
LaRankExact (online)	1195	7806
LaRankGreedyGap (batch)	940	8913
PAGreedy (online)	410	2922
LaRankGreedy (online)	905	8505

Table 5.7: **Values of dual objective after training phase.**

in time for different models. Figure 5.3 thus shows that, as we expected, our models scale linearly in time while a common batch method as SVMstruct does not.

The dual objective values reached by the online algorithms based on LaRank and by their batch counterparts are quite similar as presented on Table 5.7. LaRankExact and LaRankGreedy have good optimization abilities; they both reach a dual value close to the convergence point of their corresponding batch algorithms. We also provide the dual of PAExact and PAGreedy, the *passive-aggressive* versions (i.e. without REPROCESS) of LaRankExact and LaRankGreedy. These low values illustrate the crucial influence of REPROCESS in the optimization process, independent of the inference method.

Other Comparisons We also provide comparisons with a number of popular algorithms for both exact and greedy inference. For exact inference, the CRF results were obtained using a fast Stochastic Gradient Descent implementation[6] of Conditional Random Fields: online results were obtained after one stochastic gradient pass over the training data; batch results were measured after reaching a performance peak on a validation set. The PerceptronExact results were obtained using the structured perceptron update proposed by [Collins, 2002] and described in Section 2.2, along with the same exact inference scheme as LaRankExact. The PAExact results were obtained with the *passive-aggressive* version of LaRankExact, that is without REPROCESS or OPTIMIZE steps. For greedy inference, we report results for both PerceptronGreedy and PAGreedy. Like LaRank, these algorithms were used in a strict online setup, performing only a single pass over the training examples.

Results in Tables 5.5 and 5.6 clearly display a gap between the accuracies of these common online methods and the accuracies achieved by our two algorithms LaRankGreedy and LaRankExact. The LaRank based algorithms are the only online algorithms able to match the accuracies of the batch algorithms. Although higher than those of other online algorithms, their training times remain reasonable. For example, on our largest data set, WSJ, LaRankGreedy reaches a test set accuracy competitive with the most accurate algorithms while requiring less training time than PerceptronExact (about four milliseconds per training sequence).

[6] `http://leon.bottou.org/projects/sgd`

Results on the CHUNKING and WSJ benchmarks have been widely reported in the literature. Our CHUNKING results are competitive with the best results reported in the evaluation of the CoNLL 2000 shared task [Kudoh and Matsumoto, 2000] (F1 score 93.48). More recent works include [Zhang *et al.*, 2002] (F1 score 94.13, training time 800 seconds) and [Sha and Pereira, 2003] (F1 score 94.19, training time 5000 seconds). The Stanford Tagger [Toutanova *et al.*, 2003] reaches 97.24% accuracy on the WSJ task but requires 150,000 seconds of training. These state-of-the-art systems slightly exceed the performances reported in this work because they exploit highly engineered feature vectors. Both LaRankExact and LaRankGreedy need a fraction of these training times to achieve the full potential of our relatively simple feature vectors.

Comparing Greedy and Exact Inference

This section focuses on an empirical comparison of the differences caused by the inference schemes

Inference Cost The same training set contains more training examples for an algorithm based on a greedy inference scheme. This has a computational cost. However the training time gap between PAExact and PAGreedy in Table 5.5 and 5.6 indicates that using exact inference entails much higher computational costs because the inference procedure is more complex.

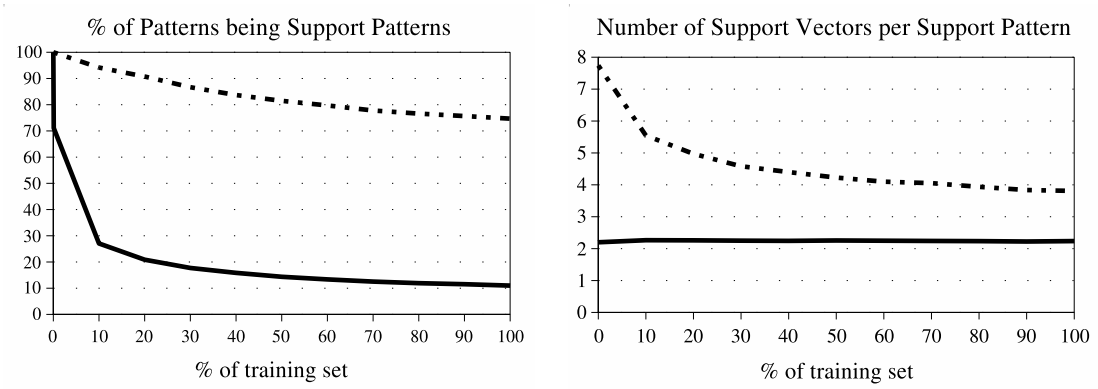


Figure 5.4: **Sparsity measures during learning on CHUNKING data set.** (Solid line: LaRankGreedy, Dashed line: LaRankExact.)

Sparsity As support vectors for LaRankExact are complete sequences, local dependencies are not represented in an invariant fashion. LaRankExact thus needs to store an important proportion of its training examples as support pattern while LaRankGreedy only requires a small fraction of them as illustrated in Figure 5.4. Moreover, for each support pattern, LaRankExact also needs to store more support vectors.

Reprocess Figure 5.5 displays the gain in test accuracy that LaRankGreedy and LaRankExact get according to the number of REPROCESS. This gain is computed relatively to the *passive-aggressive* algorithms which are similar but do not perform any REPROCESS. LaRankExact requires more REPROCESS (10 on OCR) than LaRankGreedy (only 5) to reach its best accuracy. This empirical result is verified on all three data sets. Using exact inference instead of greedy inference causes additional computations in the LaRank algorithm.

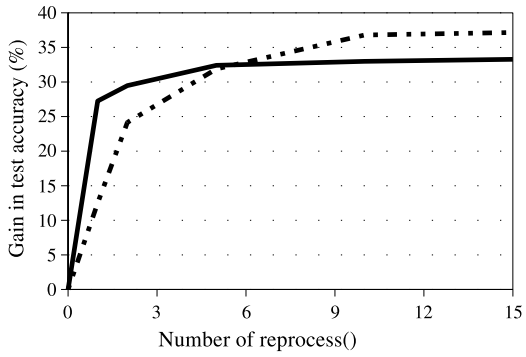


Figure 5.5: **Gain in test accuracy compared to the** *passive-aggressives* **according to n_R on** OCR**.** (Solid line: LaRankGreedy, Dashed line: LaRankExact)

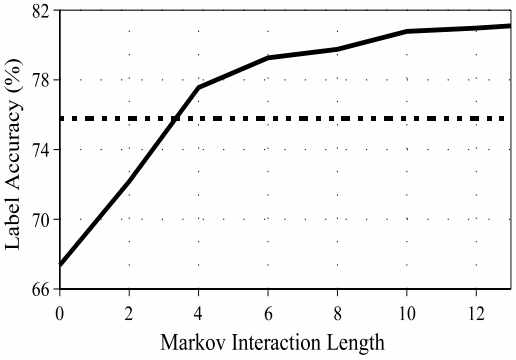


Figure 5.6: **Test accuracy according to the Markov interaction length on** OCR**.** (Solid line: LaRankGreedy, Dashed line: LaRankExact for which $k = 1$)

Markov Assumption Length This section indicates that using exact inference in our setup involves both time and sparsity penalties. Moreover the loss in accuracy that could occur when using a greedy inference process and not an exact one can be compensated by using Markov assumptions of order higher than 1. As shown on Figure 5.6 it can even lead to better generalization performances.

Wrap-up Online learning and greedy inference offer the most attractive combination of short training time, high sparsity and accuracy. Indeed, LaRankGreedy is approximately as fast as an online perceptron using exact inference, while being almost as accurate as a batch optimizer.

5.4 Summary

This chapter presented LaRank. This large-margin online learning algorithm for structured output prediction nearly reaches its optimal performance in a single pass over the training examples and matches the accuracy of batch solvers. In addition, LaRank shares the scalability properties of other online algorithms. Similarly to SVMstruct, its number of support vectors is conveniently bounded. Using an extension of [Shalev-Shwartz and Singer, 2007a] to structured outputs, we also showed that it has at least the same theoretical guarantees in terms of regret (difference between the online error and the optimal train error) as passive-aggressive algorithms.

Applied to multiclass classification and to sequence labeling, LaRank leads to empirically competitive algorithms, that learn in one epoch and reach the performance of equivalent batch algorithms on benchmark tasks. Involving low time and memory requirements, LaRank tends to be a suitable algorithm when one wants to learn structured output predictors on large-scale training data sets. We have presented two derivations but it could be applied to any structured prediction problem as soon as it can be casted in the framework described in Section 2.2. For example, [Usunier *et al.*, 2009] recently used it for learning a ranking system on large amounts of data.

6

Learning SVMs under Ambiguous Supervision

Contents

Previous chapters have presented several original supervised learning algorithms to train Support Vector Machines for a broad range of applications. Enjoying nice theoretical and experimental properties, all methods can be employed on large-scale training databases, as soon as these are annotated. Unfortunately, this last condition can be penalizing because annotating large amounts of data is often costly and time-consuming. Depending on the task, this can even require highly-advanced expertise on the part of the labeler. As we explained in Section 1.1.2, collaborative labeling or human-based computing can provide some annotations for reduced costs. However this solution can not be actually employed in any case.

Ambiguous Supervision We present here an other opportunity to bypass such costs. For many tasks an automatic use of multimodal environments can provide training corpora with little or no human processing. For instance, the time synchronisation of several media can generate annotated corpora: matching movies with subtitles [Cour *et al.*, 2008] can be used for speech recognition or information retrieval in videos, matching vision sensors and other sensors can be used to improve robotic vision (as in [Angelova *et al.*, 2007]), matching natural language and perceptive events (such as audio commentaries and soccer actions in RoboCup [Chen and Mooney, 2008]) can be used to learn semantics. Indeed, the Internet is abundant with such sources, for example one could think to use the text surrounding pictures in a webpage as image label candidates.

Such automatic procedures can build large corpora of ambiguously supervised examples. Indeed, every resulting input instance (picture, video frame, speech, ...) is paired with a set of candidate output labels (text caption, subtitle, ...). The automation of the data collection makes it impossible to directly know which one is correct among them, or even if there exists

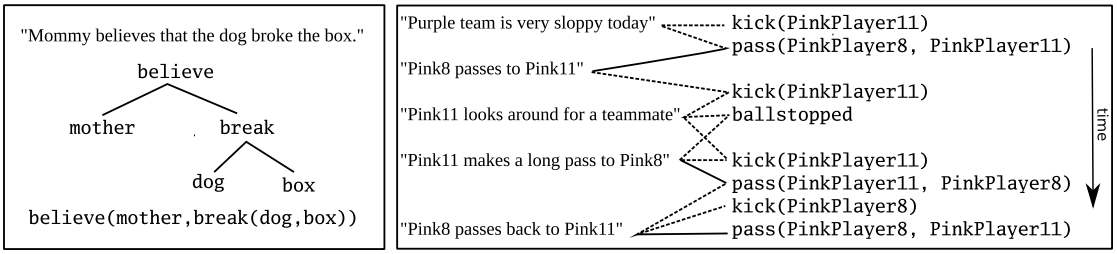


Figure 6.1: **Examples of semantic parsing.** *Left:* an input sentence (line 1) and its representation in a domain-specific formal language. *Right:* automatic generation of ambiguous training data by time-synchronisation of natural language commentaries (left column) and events in a RoboCup soccer game (right column).

a correct label. To conceive systems able to efficiently learn out of such noisy and ambiguous supervision would be a huge leap forward in machine learning. These methods could then benefit from large training sets obtained with drastically reduced costs.

Semantic Parsing In this chapter we propose to study the process of learning under ambiguous supervision through the task of semantic parsing (see e.g. [Zettlemoyer and Collins, 2005, Wong and Mooney, 2007]). This is appropriate because many of the few previous works on ambiguous supervision [Chen and Mooney, 2008, Kate and Mooney, 2007] are related to it. Semantic parsing aims at building systems that could understand questions or instructions in natural language in order to bring about a major improvement in human-computer interfacing. Formally, this consists of mapping a natural language sentence into a logical *meaning representation* (MR) which is domain-specific and directly interpretable by a computer. An example of a semantic parse is given in Figure 6.1 (left).

Semantic parsing is an interesting case study for ambiguous supervision. Indeed, the derivation from a sentence to its logical form is never directly annotated in the training data. At the word-level, semantic parsing is thus always ambiguously supervised: in the example of Figure 6.1 (left), there is no direct evidence that the word "dog" refers to the symbol `dog`. Furthermore, training data for semantic parsing can be naturally gathered within perceptive environments via the co-occurrence of language and events as in the right of Figure 6.1. Such examples are noisy and ambiguous: irrelevant actions can occur at the time a sentence is uttered, an event can be described by several sentences, and conversely a sentence can describe several events.

Ambiguously Supervised SVMs The contributions of this chapter are twofold. We first propose a reduction from multiclass classification with ambiguous supervision to noisy label ranking as well as an efficient online algorithm to solve this new formulation. We also show that, in the ambiguous learning framework, our solver has a fast decreasing regret. We then apply this algorithm to the specific case of semantic parsing. We introduce the OSPAS algorithm, a sequential method inspired by LaSO [Daumé III and Marcu, 2005]. OSPAS is able to discover the alignment between words and symbols and uses it to recover the structure of the semantic parse. Finally we provide an empirical validation of our algorithm on three data sets. First, we created a simulated data set to highlight the online ability of our method to recover the word-level alignment. We then present results on the AmbigChild-World and RoboCup semantic

parsing benchmarks on which we can compare with state-of-art semantic parsers from [Chen and Mooney, 2008, Kate and Mooney, 2007].

The rest of the chapter is organized as follows. Section 6.1 describes our general algorithm, Section 6.2 details its specialization to semantic parsing called OSAPS, Section 6.2.2 describes experimental results and Section 6.3 concludes.

6.1 Online Multiclass SVM with Ambiguous Supervision

In this section, we present the task of multiclass classification with ambiguous supervision, and justify how ambiguous supervision can be treated as a label ranking problem. We then present an efficient online procedure for training in this context.

6.1.1 Classification with Ambiguous Supervision

As in classical multiclass classification (see Section 5.2), the goal is to learn a function f that maps an observation $x \in \mathcal{X}$ to a class label $y \in \mathcal{Y}$. We still assume that f predicts the class label with a discriminant function $S(x, y) \in \mathbb{R}$ that measures the degree of association between pattern x and class label y and using a standard arg max procedure:

$$f(x) = \arg\max_{y \in \mathcal{Y}} S(x, y) \ . \tag{6.1}$$

As in previous chapters, we consider a linear form for $S(x, y)$ i.e. $S(x, y) = \langle w, \Phi(x, y) \rangle$, where $\Phi(x, y)$ maps the pair (x, y) into a feature space endowed with the dot product $\langle \cdot, \cdot \rangle$, and w is a parameter vector to be learnt.

Given a class of functions $\mathcal{F}$, we consider an *ambiguous supervision* setting, where a training instance $(x, \mathbf{y})$ consists of an observation $x \in \mathcal{X}$ and a set $\mathbf{y} \in \mathcal{P}(\mathcal{Y}) \setminus \mathcal{Y}$ of class labels, where $\mathcal{P}(\mathcal{Y})$ is the power set of $\mathcal{Y}$.[1] The semantics of this set $\mathbf{y}$ is that at least one of class labels present in the set $\mathbf{y}$ should be considered as the *correct* class label of x (i.e. the one that should be predicted), but some of the class labels in $\mathbf{y}$ might not be correct. We define this particular label using the following function:

$$y^*(x) = \arg\max_{y \in \mathcal{Y}} \mathbb{P}_{\mathbf{y}}(y \in \mathbf{y} | x) \ . \tag{6.2}$$

Hence, assuming that the observations are drawn according to a fixed distribution $\mathcal{D}$ on $\mathcal{X}$, we expect the prediction function f to minimize the following error:

$$\mathrm{err}^*(f) = \mathbb{P}_{x \sim \mathcal{D}}(f(x) \neq y^*(x)) \ . \tag{6.3}$$

Related Work

[Cour *et al.*, 2009] recently proposed to solve the problem of learning under ambiguous supervision with a slightly different approach. They employ the pointwise error of a multiclass classifier f on an ambiguous example $(x, \mathbf{y})$, defined as:

$$\mathrm{err}^{0/1}(f, (x, \mathbf{y})) = \mathrm{I}(f(x) \notin \mathbf{y}) = \mathrm{I}\left(\forall y \in \mathbf{y}, \exists \bar{y} \in \mathcal{Y} \backslash \mathbf{y}, S(x, y) < S(x, \bar{y})\right).$$

Then, they showed, under natural assumptions on the nature of the ambiguities, that the minimizers of $\mathbb{E}\left[\mathrm{err}^{0/1}(f, (x, \mathbf{y})\right]$ are close to those of the unambiguous case. Thus, they tackle the

[1] We obviously require that the supervision does not consist of the whole set of class labels, since the example is uninformative in that case.

ambiguity by considering an unambiguous error different from err^*. Yet both errors track the same optimal prediction y^* but give different guarantees.

Unfortunately, $\text{err}^{0/1}$ is difficult to deal with because it naturally leads to non-convex optimization problems. For instance, if we consider the linear and realizable case, a natural large-margin formulation that corresponds to this error on a training set $(x_i, \mathbf{y}_i)_{i=1}^n$ is:

$$\min_w \quad \frac{1}{2}||w||^2 \quad u.c. \quad \forall i, \exists y \in \mathbf{y}_i \text{ such that } \forall \bar{y} \in \mathcal{Y} \backslash \mathbf{y}_i \quad \langle w, \Phi(x_i, y) \rangle - \langle w, \Phi(x_i, \bar{y}) \rangle \geq 1. \tag{6.4}$$

Even if this problem is feasible, its optimization rapidly becomes intractable, since it is highly non-convex due to the existential quantifier in the constraints. [Cour *et al.*, 2009] proposed a convex upper bound on $\text{err}^{0/1}$, which reduces to the *One-Versus-All* approach to multiclass classification in the unambiguous case but, they did not exhibit assumptions sufficient to guarantee that minimizing this error (or some 0/1 version of it) allows to recover the correct labels.

Reduction to Label Ranking

In our method, to find a minimizer of err^*, we propose to follow a label ranking approach which, in the unambiguous case, boils down to the constraint classification approach of [Har-Peled *et al.*, 2002]. We thus use the mean pairwise error:

$$\text{err}^p(f, (x, \mathbf{y})) = \frac{1}{|\mathbf{y}||\mathcal{Y} \backslash \mathbf{y}|} \sum_{y \in \mathbf{y}} \sum_{\bar{y} \in \mathcal{Y} \backslash \mathbf{y}} \left(\mathrm{I}\left(S(x, y) < S(x, \bar{y})\right) + \frac{1}{2} \mathrm{I}\left(S(x, y) = S(x, \bar{y})\right) \right)$$

In the case of unambiguous multiclass classification, linear multiclass classifiers may be learnable in the constraint classification setting, but not in the One-versus-All one (see Section 3.3 of [Har-Peled *et al.*, 2002]). Moreover, we show in the next section that, under natural assumptions, minimizing the mean pairwise error err^p allows to minimize err^* and recover the correct labels.

In terms of optimization procedure, the mean pairwise error in the case of linear functions can be optimized on a training set $(x_i, \mathbf{y}_i)_{i=1}^m$ with the following standard soft-margin SVM formulation ($[t]_+$ denotes the positive part of t):

$$\min_w \frac{1}{2}||w||^2 + C \sum_{i=1}^m L_w(x_i, \mathbf{y}_i) \tag{6.5}$$

where $L_w(x_i, \mathbf{y}_i) = \dfrac{1}{|\mathbf{y}_i||\mathcal{Y} \backslash \mathbf{y}_i|} \displaystyle\sum_{y \in \mathbf{y}_i} \sum_{\bar{y} \in \mathcal{Y} \backslash \mathbf{y}_i} \left[1 - \langle w, \Phi(x_i, y) - \Phi(x_i, \bar{y}) \rangle\right]_+ .$

Unbiased Ambiguity

We now formally justify the use of the mean pairwise loss as a possible alternative for multiclass learning with ambiguous supervision: under the following assumptions, we show that if the incorrect labels given as supervision are random given the input x then err^p has the same minimizer on any distribution on the input space as err^*, even in the presence of random noise (i.e. the correct label is not given).

For simplicity, we assume that for any observation x, the set $\mathbf{y}$ given as supervision is of constant length. We consider the setting where the *correct label* of any input observation is given by the function $y^* \in \mathcal{F}$. That is, the target classifier is in the class of hypotheses. We also make the three following natural assumptions:

1. $\forall y, y' \neq y^*(x) \;\; P(y \in \mathbf{y}|x) = P(y' \in \mathbf{y}|x)$,
2. $\exists \gamma > 0, \forall x, \; P(y^*(x) \in \mathbf{y}|x) > P(y \in \mathbf{y}|x) + \gamma$ for $y \neq y^*(x)$,

3. $\forall y \neq y^*(x)\,,\ S^*(x, y^*(x)) > S^*(x, y)$ with S^* the score function associated with y^*.

The first assumption is the *unbiased ambiguity* assumption which ensures that the distribution of incorrect labels within the supervision bags is not biased towards any incorrect label. The second one forces the correct labels to appear in the supervision more often than the incorrect ones. But it does not forbid cases where the correct label is not given in the supervision. The third one makes sure that the argmax equation (6.1) always defines a single label.

Then, the following theorem holds. We provide a result for err^* in the general i.i.d. case but also a result in the non i.i.d. case because this is useful in an online setup. In the non i.i.d. case, the error to minimize is defined as $\frac{1}{n}\sum_{i=1}^{n} \mathrm{I}\left(f(x_i) \neq y^*(x_i)\right)$ for n observations, $x_1, ..., x_n$.

Theorem 12 *Under the previous assumptions, we have:*

I.i.d. case. *Assume the observations are drawn according to a fixed distribution $\mathcal{D}$ on $\mathcal{X}$. Then, for all $f \in \mathcal{F}$:*

$$err^*(f) \leq \frac{2\ell(|\mathcal{Y}| - \ell)}{\gamma}\, \mathbb{E}\left[err^p(f, (x, \mathbf{y})) - err^p(y^*, (x, \mathbf{y}))\right]$$

where ℓ is the size of the ambiguous supervision sets, and the expectations are taken for $x \sim \mathcal{D}$ and $\mathbf{y} \sim P(.|x)$

Non-i.i.d. case *Let $x_1, ..., x_n$ be n observations. Then, for all $f \in \mathcal{F}$:*

$$\frac{1}{n}\sum_{i=1}^{n} \mathrm{I}\left(f(x_i) \neq y^*(x_i)\right) \leq \frac{2\ell(|\mathcal{Y}| - \ell)}{\gamma}\, \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(err^p(f, (x_i, \mathbf{y}_i)) - err^p(y^*, (x_i, \mathbf{y}_i))\right)\right]$$

where the expectations are taken over $\mathbf{y}_i \sim P(.|x_i)$.

Proof Both proofs follow from a direct calculation of $\mathbb{E}\left[\text{err}^p(f, (x, \mathbf{y}))|x\right]$, the expectation of the pairwise error of f on a fixed observation x. Following the definition of the mean pairwise error, we have:

$$\begin{aligned}
\mathbb{E}\left[\text{err}^p(f, (x, \mathbf{y}))|x\right] &= \sum_{\mathbf{y}} \frac{P(\mathbf{y}|x)}{\ell(|\mathcal{Y}| - \ell)} \sum_{y \in \mathbf{y}} \sum_{\bar{y} \in \mathcal{Y} \setminus \mathbf{y}} s(x, y, \bar{y}) \\
&= \frac{1}{\ell(|\mathcal{Y}| - \ell)} \sum_{y \in \mathcal{Y}} \sum_{\bar{y} \in \mathcal{Y}} P(y \in \mathbf{y}, \bar{y} \notin \mathbf{y}|x) s(x, y, \bar{y})
\end{aligned}$$

where $s(x, y, \bar{y}) = \mathrm{I}\left(S(x, y) < S(x, \bar{y})\right) + \frac{1}{2}\mathrm{I}\left(S(x, y) = S(x, \bar{y})\right)$.

Using the assumption $P(y \in \mathbf{y}|x) = P(y' \in \mathbf{y}|x)$ for any $y, y' \neq y^*(x)$, and by elementary probability calculus, we have $P(y \in \mathbf{y}, y' \notin \mathbf{y}|x) = P(y' \in \mathbf{y}, y \notin \mathbf{y}|x)$. Grouping the corresponding two terms in the sum and noticing that $s(x, y, \bar{y}) + s(x, \bar{y}, y) = 1$, we obtain:

$$\begin{aligned}
\mathbb{E}\left[\text{err}^p(f, (x, \mathbf{y}))|x\right] =& \frac{1}{2\ell(|\mathcal{Y}| - \ell)} \sum_{y \neq y^*(x)} \sum_{\bar{y} \neq y^*(x)} P(y \in \mathbf{y}, y' \notin \mathbf{y}|x) \\
&+ \frac{1}{\ell(|\mathcal{Y}| - \ell)} \sum_{y \in \mathcal{Y}} \Big[P(y \in \mathbf{y}, y^*(x) \notin \mathbf{y}|x) s(x, y, y^*(x)) \\
&\qquad + P(y^*(x) \in \mathbf{y}, y \notin \mathbf{y}|x) s(x, y^*(x), y)\Big]
\end{aligned}$$

The first term is constant over all f. With the same calculation for the specific y^*, we can notice that, (1) if S^* is the discriminant function associated to y^* (i.e. $S^*(x, y) = \langle w^*, \Phi(x, y)\rangle$), $s^*(x, y, y^*(x)) = 0$, (2) $P(y^*(x) \in \mathbf{y}, y^*(x) \notin \mathbf{y}|x) = 0$, and (3) for any y, $P(y^*(x) \in \mathbf{y}, y \notin \mathbf{y}|x) - P(y \in \mathbf{y}, y^*(x) \notin \mathbf{y}|x) = P(y^*(x) \in \mathbf{y}|x) - P(y \in \mathbf{y}|x)$. We finally obtain:

$$\mathbb{E}\left[\text{err}^p(f, (x, \mathbf{y})) - \text{err}^p(y^*, (x, \mathbf{y}))|x\right] = \frac{P(y^*(x) \in \mathbf{y}|x) - p}{\ell(|\mathcal{Y}| - \ell)} \sum_{y \neq y^*(x)} s(x, y^*(x), y)$$

where $p = P(y \in \mathbf{y}|x)$ for any $y \neq y^*(x)$. Since $f(x) \neq y^*(x)$ as soon as $\sum_{y \neq y^*(x)} s(x, f(x), y) > 0$ and that this sum is always greater than $1/2$ when strictly positive, we have both desired results (the first one by taking the expectation over x, the second by summing over the n given $x_1, ..., x_n$). □

Interpretation In the i.i.d. setting, the theorem shows that any minimizer in $\mathcal{F}$ of the true (i.e. generalization) pairwise loss recovers the function that produces the correct labels (and that y^* minimizes the pairwise loss in $\mathcal{F}$). Since it can be shown (e.g. pursuing a growth function approach as in [Har-Peled *et al.*, 2002]) that the minimizer of the empirical risk in the mean pairwise setting converges to the minimizer of the true risk, this justifies the use of the mean pairwise loss in the ambiguous classification setting.

When the observations are fixed we have a similar result. We provide this version since we will use the pairwise loss in the online setting, where the data may not be i.i.d. The results are interesting in terms of regret, because an algorithm with a regret (in terms of pairwise loss) that converges to zero corresponds to an algorithm which predicts the correct label up to some point.

6.1.2 Online Algorithm

There has been a lot of work on online algorithms for label ranking (see e.g. [Crammer and Singer, 2005, Crammer *et al.*, 2006, Shalev-Shwartz and Singer, 2007b]). We present here an algorithm that follows the primal-dual perspective presented in [Shalev-Shwartz and Singer, 2007a], and can be seen as an analog of the algorithm of [Shalev-Shwartz and Singer, 2007b] (which uses the maximum pairwise loss) for the mean pairwise loss.

The algorithm is based on a formulation of the SVM primal problem (6.5) using a single slack variable per example. Using the equality $[1-t]_+ = \max_{c \in \{0,1\}} c(1-t)$, the mean pairwise hinge loss on a given example $(x_i, \mathbf{y}_i)$ can be written as:

$$\begin{aligned} L_w(x_i, \mathbf{y}_i) &= \max_{\mathbf{c}} \frac{1}{|\mathbf{y}_i||\mathcal{Y}\backslash\mathbf{y}_i|} \sum_{y \in \mathbf{y}_i} \sum_{\bar{y} \in \mathcal{Y}\backslash\mathbf{y}_i} c_{y\bar{y}} \left(1 - \langle w, \Phi(x_i, y) - \Phi(x_i, \bar{y})\rangle\right) \\ &= \max_{\mathbf{c}} \left(\Delta_{x_i,\mathbf{y}_i}(\mathbf{c}) - \langle w, \Psi_{x_i,\mathbf{y}_i}(\mathbf{c})\rangle\right) \end{aligned} \tag{6.6}$$

with $\mathbf{c} \in \{0,1\}^{|\mathbf{y}_i||\mathcal{Y}\backslash\mathbf{y}_i|}$, and

$$\begin{aligned} \Delta_{x_i,\mathbf{y}_i}(\mathbf{c}) &= \frac{1}{|\mathbf{y}_i||\mathcal{Y}\backslash\mathbf{y}_i|} \sum_{y \in \mathbf{y}_i} \sum_{\bar{y} \in \mathcal{Y}\backslash\mathbf{y}_i} c_{y\bar{y}} \\ \Psi_{x_i,\mathbf{y}_i}(\mathbf{c}) &= \frac{1}{|\mathbf{y}_i||\mathcal{Y}\backslash\mathbf{y}_i|} \sum_{y \in \mathbf{y}_i} \sum_{\bar{y} \in \mathcal{Y}\backslash\mathbf{y}_i} c_{y\bar{y}} \left(\Phi(x_i, y) - \Phi(x_i, \bar{y})\right). \end{aligned} \tag{6.7}$$

This leads to the SVM primal formulation:

$$\begin{aligned} &\min_{w,\xi} \frac{1}{2}||w||^2 + C\sum_i \xi_i \\ &\text{subject to} \begin{cases} \forall i \;\; \xi_i \geq 0 \\ \forall i \;\; \forall \mathbf{c} \in \{0,1\}^{|\mathbf{y}_i||\mathcal{Y}\backslash\mathbf{y}_i|} \; \langle w, \Psi_{x_i,\mathbf{y}_i}(\mathbf{c})\rangle \geq \Delta_{x_i,\mathbf{y}_i}(\mathbf{c}) - \xi_i \end{cases} \end{aligned} \tag{6.8}$$

Our algorithm optimizes the dual of (6.8):

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_{i,\mathbf{c}} \alpha_i^{\mathbf{c}} \Delta_{x_i,\mathbf{y}_i}(\mathbf{c}) - \frac{1}{2} \sum_{i,\mathbf{c}} \sum_{j,\bar{\mathbf{c}}} \alpha_i^{\mathbf{c}} \alpha_j^{\bar{\mathbf{c}}} \left\langle \Psi_{x_i,\mathbf{y}_i}(\mathbf{c}), \Psi_{x_j,\mathbf{y}_j}(\bar{\mathbf{c}})\right\rangle.$$

Following [Shalev-Shwartz and Singer, 2007a], an online algorithm can be derived from the dual function using a simple dual coordinate ascent procedure in a passive-aggressive setup. While iterating over the examples, a single parameter update is performed for each example using a dual coordinate associated to the given example and the step size that maximizes the dual increase. A box constraint enforces the step size to remain between 0 and C.

Algorithm 23 AMBIGSVMDUALSTEP

1: **input:** $x_t \in \mathcal{X}$, $\mathbf{y}_t$.
2: Get $\Delta_{x_t,\mathbf{y}_t}(\mathbf{c}_t), \Psi_{x_t,\mathbf{y}_t}(\mathbf{c}_t)$ where $\Delta_{x_t,\mathbf{y}_t}(\mathbf{c}_t) - \langle w, \Psi_{x_t,\mathbf{y}_t}(\mathbf{c}_t)\rangle = L_w(x_t, \mathbf{y}_t)$
3: Compute $\alpha_t^{\mathbf{c}_t} = \frac{\Delta_{x_t,\mathbf{y}_t}(\mathbf{c}_t) - \langle w, \Psi_{x_t,\mathbf{y}_t}(\mathbf{c}_t)\rangle}{||\Psi_{x_t,\mathbf{y}_t}(\mathbf{c}_t)||^2}$
4: Clip $\alpha_t^{\mathbf{c}_t} = \max(0, \min(\alpha_t^{\mathbf{c}_t}, C))$
5: Update $w = w + \alpha_t^{\mathbf{c}_t} \Psi_{x_t,\mathbf{y}_t}(\mathbf{c}_t)$

Algorithm 23 summarizes the steps followed by the algorithm when it receives a new example $(x_t, \mathbf{y}_t)$. In our setting, after having seen the t-th example, the chosen dual coordinate is $\alpha_t^{\mathbf{c}_t}$ (line 2), with $\mathbf{c}_t$ the binary vector that realizes the max of equation (6.6). The value given to this dual variable is computed analytically by maximizing the dual along the chosen direction (line 3) and clipping it to the constraint (line 4). The parameter vector w is finally updated (line 5).

Regret Bound

Following [Shalev-Shwartz and Singer, 2007a] and the work presented in Chapter 5, the generalization ability of an online algorithm sequentially increasing the dual objective function can be expressed in terms of regret. The regret is defined by the difference between the mean loss incurred by the algorithm on the course of learning and the empirical loss of a given weight vector w that is, $\text{regret}(n, w) = \frac{1}{n}\sum_{i=1}^{n} L_{w_i}(x_i, \mathbf{y}_i) - \frac{1}{n}\sum_{i=1}^{n} L_w(x_i, \mathbf{y}_i)$ with w_i the parameter vector before seeing the i-th example.

Proposition 13 *Define* $\rho = \max_{i, y\in\mathbf{y}_i, \bar{y}\in\mathcal{Y}\setminus\mathbf{y}_i} ||\Phi(x_i, y) - \Phi(x_i, \bar{y})||^2$. *After seeing* n *examples, the regret of Algorithm 23 is upper-bounded:* $\forall w,\ \text{regret}(n, w) \leq \frac{||w||^2}{2nC} + \frac{\rho C}{2}$.
Furthermore, if $C = \sqrt{\frac{||w||^2}{n\rho}}$ *then:* $\forall w,\ \text{regret}(n, w) \leq \sqrt{\frac{\rho||w||^2}{n}}$.

This proposition, easily established by directly following the proof of Theorem 10 of Section 5.1.5, exhibits that the regret of the online multiclass SVM for ambiguous supervision has the compelling property of decreasing with the number of training examples.

6.2 Sequential Semantic Parser

The previous section defined an algorithm for learning multiclass classification under ambiguous supervision. In order to benchmark it, we now use it for learning semantic parsing under ambiguous supervision.

6.2.1 The OSPAS Algorithm

This section describes how we applied the online SVM (Algorithm 23) to derive an algorithm for semantic parsing.

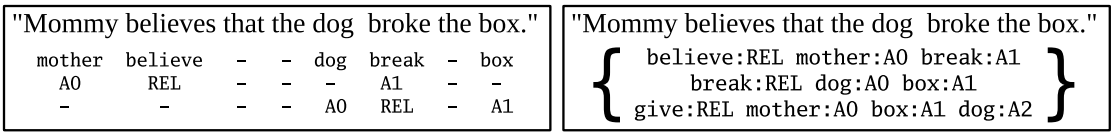


Figure 6.2: **Semantic parsing training example.** *Left:* Predicted parse. Words are successively labeled with symbols (line 2) and SRL tags (line 3-4). *Right:* Training example. Several MRs are given in supervision: a combination of them can represent the correct MR (line 2-3), some might not be related (line 4). Empty label pairs (-,-) are also added to the bag.

Predicted Meaning Representations

The MRs considered in semantic parsing are simple logical expressions of the form $REL(A0, A1, \ldots, An)$. REL is the relation symbol, and $A0$, ..., An its arguments. Notice that several forms can be recursively constructed to form more complex tree structures.[2] For instance, the tree in Figure 6.1 (left) is equivalent to the representation given in Figure 6.2 (left).

In our work, we consider the latter equivalent representation of the MRs which allows, for a given sentence, to create the semantic parse in several tagging steps. The first step is called *symbol labeling*, and consists in labeling each word of a sentence with the its corresponding symbol in the MR. This step is followed by *semantic role labeling* (SRL) steps: for each predicted relation symbol, its arguments are labeled.

The crucial feature of this alternative representation is the use of the alignment word-symbol. This can be seen as a nice way of encoding the joint structure of the sentence and the MRs and this allows to predict the final MRs in several distinct steps. This is simpler than a global joint inference step over the sentence and the MRs tree.

The *ambiguous* supervision consists of providing several MRs for each training sentence: it is unknown which is the correct MR or combination of MRs. An example of a training instance is given in Figure 6.2 (right). For our training algorithm the available supervision consists in the pairs (Symbol, SRL tag) that appear in the different MRs. As an alignment word-symbol must be feasible for each MRs, the supervision is completed with empty label pairs (-,-) if the number of symbols in the MRs is lower than the length of the input sentence. We refer to this supervision as a *bag* of pairs as it can contain duplicates of the same symbol.

The OSPAS Algorithm

We now describe OSPAS, the *Online Semantic Parser for Ambiguous Supervision*. Presented in Algorithm 24, it is firstly designed to perform the symbol prediction step. Taking as input a sentence $\mathbf{x}$ it follows the LaSO algorithm [Daumé III and Marcu, 2005] by incrementally building the output sequence. Each atomic symbol is predicted using Algorithm 23: this is the base classifier that can learn with the ambiguously supervised semantic parsing data.

For training, OSPAS receives a bag of symbols $\mathbf{b}$. At each training step, an unlabeled word of the sentence is *randomly* picked (line 6) to tend to satisfy the random ambiguity assumption (see Section 6.1.1). If the corresponding predicted label violates the supervision (not in the bag – line 8), an update of Algorithm 23 is performed (line 9). The word is removed from the unlabeled

[2] In our work, we do not use any hard-coded grammar nor decoding step during parsing, because we do not need to. The approach can however be adapted to use a grammar and a global inference procedure for predicting the parse tree as soon as the symbols have been detected and aligned.

Algorithm 24 OSPAS. `choose`(s) randomly samples without replacement in the set s and `bagtoset`(b) returns a set after removing the redundant elements of b.

```
1: input: A sentence x = (x_1, ...., x_|x|) and a bag b = {y_1, ..., y_|b|}.
2: Initialize the set UNLABELED = {x_1, ...., x_|x|};
3: while |UNLABELED| > 0 do
4:     Set s_0 = |UNLABELED|
5:     for i = 1, ..., s_0 do
6:         x_k = choose(UNLABELED);
7:         ŷ = arg max_{y∈Y} S(x_k, y);
8:         if ŷ ∉ b then
9:             Perform an update: AmbigSVMDualStep(x_k, bagtoset(b));
10:        else
11:            Remove ŷ from b and x_k from UNLABELED;
12:            break;
13:        end if
14:    end for
15:    if |UNLABELED| = s_0 then
16:        break;
17:    end if
18: end while
```

set only if the prediction was in the bag (line 11): this enforces the SVM to perform a lot of updates, especially at the beginning of training.

A crucial point of OSPAS is the bag management. Indeed if the bag was kept fixed during all the predictions on a sentence, nothing would forbid the empty symbol "-" to be predicted for every word of the sentence: it would never violate the supervision as it is added to almost every training bag. To prevent such trivial (and incorrect) solutions, we remove a symbol from the bag as soon as it has been predicted (line 11).

Specific Learning Setup

Our feature system is very simple.[3] Each word $x \in \mathcal{X}$ is encoded using a "window" representation: $x = (C(i-l), \ldots, C(i+l))$, where $C(j)$ is a binary vector with as many components as there are words in the vocabulary, and all components are set to 0 except the one that corresponds to the j-th word of the input sentence. $\Phi(x, y)$ is also binary vector of size $\mathcal{X} \times \mathcal{Y}$: only the features associated with symbol y can be non-zero.

For symbol prediction, a window of size 1 is sufficiently informative. Therefore, if we set $C = \sqrt{\frac{|\mathcal{X}||\mathcal{Y}|}{2n}}$, a direct analytical calculation under this simple feature setup can drastically simplify the bound of Proposition 13. The regret of Algorithm 23 w.r.t. a parameter vector w^* minimizing the primal (6.8) is now upper-bounded by $\sqrt{2|\mathcal{X}||\mathcal{Y}|/n}$. The regret decreases very fast with n: this can explain why OSPAS reaches good accuracies after a single pass over the data[4] (see Section 6.2.2).

Given an input sentence, OSPAS outputs a symbol sequence aligned with it. To finally recover the MR, one has to perform as many SRL tagging steps as there are `RELs` in the predicted symbols (we assume we know which symbols are `REL`). As the bag of supervision provides the corresponding SRL tag for each symbol, OSPAS can also be used to learn the SRLs with the

[3] This is a basic setup, and it would be easy to add part-of-speech, chunk or parse tree based features.

[4] We recall that, in the regret bound, n refers to the number of *words* seen by the algorithm.

sentence and the aligned symbols as input. We need to refine the feature representation by using a larger input window.

The global system is trained online. Given an input sequence and its bag (symbols, SRLs) a first OSPAS model learns the symbol prediction and a second one the SRL tagging. For simplicity reasons, we will refer to the whole system as OSPAS in the following section.

6.2.2 Experiments

Training a semantic parser in a real perceptive environment is challenging as it would require a process of generating *meaning representations* out of real-world perception data, using advanced technologies such as visual scene understanding. We thus empirically assess our approach on two benchmarks and a toy data set.

Experimental Setup

The first benchmark is AmbigChild-World from [Kate and Mooney, 2007]. It has been conceived to mimic the type of language data that would be available to a child while learning a language. The corpus is generated to model occasional language commentary on a series of perceptual contexts. A synchronous context-free grammar generates natural language sentences and their corresponding MRs which are in predicate logic without quantification, as illustrated in the example of Figure 6.2 (right). The generated MRs can be quite complex, containing from one to four `RELs`. The data set contains ten splits of 900 training instances and 25 testing one. We present results averaged on these ten splits.

The RoboCup commentary benchmark contains human commentaries on football simulations over four games labeled with semantic descriptions of actions (passes, offside, ...) and is composed of pairs of commentaries and actions that occurred within 5 seconds of each other. Following [Chen and Mooney, 2008] we trained on three games and tested on the last one, averaging over all four possible splits. This leads to an average of 1,200 training instances and 400 testing ones. This data set is ambiguous and also very noisy: around 30% of the supervision bags do not even contain the correct MRs.

To assess the ability of our method to recover the correct word-level alignment we needed a data set for which such an alignment exists. Following [Kate and Mooney, 2007] we created a simulation of a house with actors, objects and locations, and generated natural languages sentences and their corresponding MRs using a simple grammar. The perfect noise-free word-symbol alignment was employed in test to evaluate the symbol predictor (but never for prediction nor training). There are 15 available actions for a total of 59 symbols and 74 words in the dictionary. We use 2,000 training sentences and 500 for test.

For AmbigChild-World and AmbigHouse the level of ambiguity in the supervision can be controlled. An ambiguity of *level* n means that, on average, supervision bags contain the correct MRs and n incorrect ones. For *no ambiguity*, only the correct MRs are given. RoboCup is naturally ambiguous but an unambiguous version of each game is provided for evaluation, containing commentaries manually associated with their correct MRs (if any). For each data set the test examples are composed by a sentence and its corresponding correct MRs only. The values of the C parameter for OSPAS are set using the online regret. We used $C = 0.1$ on AmbigHouse and RoboCup, and $C = 1$ on AmbigChild-World (the OSPAS models for symbol and SRL predictions use the same C). All results presented for OSPAS have been obtained *after a single pass* of an online training on the data.

The main baselines for ambiguous semantic parsing are KRISPER [Kate and Mooney, 2007] and WASPER [Chen and Mooney, 2008]. Both methods follow the same training process. They

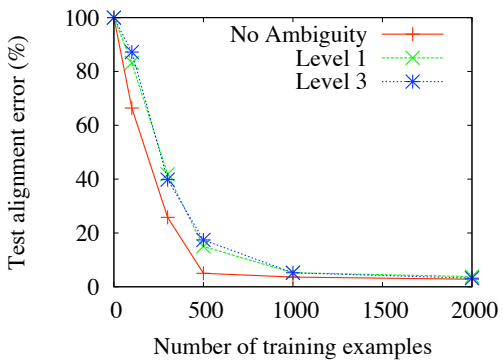


Figure 6.3: **Online test error curves on** AMBIGHOUSE for different levels of ambiguity. (Only one online training epoch.)

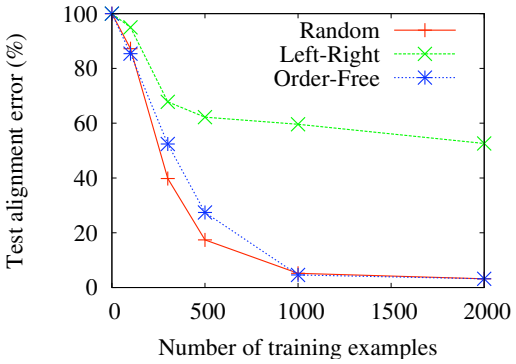


Figure 6.4: **Influence of the exploration strategy on** AMBIGHOUSE for an ambiguity of level 3. (Only one online training epoch.)

build noisy, unambiguous data sets from an ambiguous one, and then train a parser designed for unambiguous supervision only (resp. KRISP [Kate and Mooney, 2006] and WASP [Wong and Mooney, 2007]). Initially, the unambiguous data set consists of all (sentence, MR) pairs occurring in the ambiguous data set. A trained classifier (initially trained as if all pairs are correct) is iteratively used to predict which of the *ambiguous* pairs are correct, the others are down-weighted (or not used) in the next round. OSPAS is more flexible: it learns in one pass and avoids costly iterative re-training phases and does not rely on any reduction to unambiguous supervision.

Results

Figure 6.3 presents the test alignment error of OSPAS according to the number of training examples for different levels of ambiguity. The alignment error is defined as the percentage of sentences for which the predicted symbol sequence is either incorrect or misaligned. This figure demonstrates that OSPAS can recover the correct alignment with an online training and even with an highly ambiguous supervision. When the ambiguity level increases, OSPAS still achieves a good accuracy, it only requires more training examples.

Figure 6.4 demonstrates that OSPAS can deal with ambiguous supervision regardless of its inference process. Indeed, for an ambiguity of level 3, we compare three strategies:

- Random: the next word to tag is selected randomly in the set UNLABELED (this is default strategy implemented by the `choose`() function of training Algorithm 24);
- Left-Right: the next word to tag is right next to the current one;
- Order-Free: all the remaining words in the set UNLABELED are tagged using step 7 of Algorithm 24. Only the prediction achieving the highest score is kept.

For all strategies, the test error decreases on the course of training. Yet, inference influences learning speed and Left-Right strategy appears to be penalized.

Tables 6.1 and 6.2 respectively present the results on AMBIGCHILD-WORLD and ROBOCUP and allow to compare OSPAS with previously published semantic parsers. The metric we used

Ambiguity Level	F1-score krisper	F1-score ospas
None	**0.940***	**0.940**
1	**0.935***	0.926
2	0.895*	**0.912**
3	0.815*	**0.891**

Table 6.1: **Semantic parsing F1-scores on** AMBIGCHILD-WORLD**.** (*)Values reproduced from [Kate and Mooney, 2007]

Method	Ambiguity	F1-score
wasp	No	0.780†
ospas	No	*0.871*
wasper	Yes	0.545†
krisper	Yes	**0.740**†
ospas	Yes	**0.737**

Table 6.2: **Semantic parsing F1-scores on** ROBOCUP**.** (†)Values reproduced from [Chen and Mooney, 2008]

is the one usually employed to evaluate semantic parsers (e.g. in [Chen and Mooney, 2008, Kate and Mooney, 2007, Zettlemoyer and Collins, 2005]): the F1-score, defined as the harmonic mean of precision and recall. In this setup, precision is the fraction of the valid MRs (i.e. conform to the MR grammar) outputted by the system that are correct and recall is the fraction of the MRs from the test set that the system correctly produces. The results on AMBIGCHILD-WORLD and ROBOCUP express that, in spite of its simple learning process and its single pass over the training data, OSPAS reaches state-of-the-art F1-scores. Indeed, it is equivalent to KRISPER and much better than WASPER. In particular, it is worth noting that OSPAS efficiently handles the high level of noise of the natural language sentences of ROBOCUP. Finally, Table 6.1 shows that OSPAS is more robust to the increase of the ambiguity level than KRISPER.

6.3 Summary

This chapter studied a novel problem of learning from ambiguous supervision focusing on the case of semantic parsing. This problem is original and interesting as ambiguous supervision issue might be crucial in the next few years.

We proposed an original reduction from multiclass classification with ambiguous supervision to noisy label ranking and derived an online algorithm for semantic parsing. Our approach is competitive with state-of-the-art semantic parsers after a single pass over ambiguously supervised data and would then hopefully scale well on future larger corpora.

7

Conclusion

Contents

This final chapter is destined to summarize our contributions but also to explain how we think they can be pursued. In a first section, we highlight our main achievements and display some straightforward extensions that could be carried out without much difficulty. In a second time, we present some artificial intelligence issues which, we conjecture, could be addressed by some derivations of the work contained by this dissertation.

7.1 Large Scale Perspectives for SVMs

Throughout this thesis, we have exhibited several ways to handle large-scale data with Support Vector Machines. For different kinds of data, different kinds of tasks, different kinds of kernels, we have proposed solutions to reduce training time and memory requirements while keeping accurate generalization performances.

Chapter 3 presented the specific issue of Stochastic Gradient Descent algorithms for learning linear SVMs and proposed the new SGD-QN algorithm. Chapter 4 explained the original Process/Reprocess principle via the simple Huller algorithm and analyzed the fast and efficient LaSVM algorithm for solving binary classification. It also investigated the benefit of joining active and online learning and defined a fresh duality lemma for incremental SVM solvers. In Chapter 5, we presented the fourth new algorithm of this dissertation: LaRank an algorithm implementing the Process/Reprocess principle to learn SVMs for structured output prediction. We detailed and tested specific derivations to multiclass classification and sequence labeling. Finally we introduced the original framework of learning under ambiguous supervision in Chapter 6 and applied it to the Natural Language Processing problem of semantic parsing which aims at building systems that could interpret instructions in natural language.

7.1.1 Impact and Limitations of our Contributions

This dissertation encompasses several new algorithms for learning large scale Support Vector Machines. Most of our work have been spread in the machine learning scientific community via international publications and talks (see personal bibliography in Appendix A) and have had a significant impact. For instance, LaSVM is now a standard method for learning SVMs and has been used as a reference in many publications.

Our contributions are composed as a mix of algorithms and implementation, and are designed towards efficiency on large-scale databases. Hence, a crucial part of our work consists in the extensive empirical validations of our methods. Let us briefly highlight some of their most remarkable experimental achievements.

- SGD-QN won the first Pascal Large Scale Learning Challenge ("Wild track") (Section 3.2.3);
- LaSVM (Section 4.2) has been successfully trained on 8.1 millions examples on a single CPU with 6.5 GB of RAM [Loosli *et al.*, 2007] (might be a world record);
- For sequence labeling, LaRank enjoys a linear scaling of time w.r.t. training set size (Section 5.3.4);
- LaRankGreedy is as fast as a perceptron and as accurate as a batch method (Section 5.3.4).

Moreover, we also presented an essential theoretical tool for large-scale training methods. Indeed, the lemma presented in Section 4.4 is critical because it provides generalization guarantees for incremental learning algorithms without requiring any additional cost.

We finally want to point out that this thesis contains one of the very first work on learning under ambiguous supervision with [Kate and Mooney, 2007] and [Cour *et al.*, 2009]. We thus cast a light on a fresh issue that might gain a growing importance in the next few years.

Throughout this thesis, our main motivation has been to propose algorithms to train SVMs on possibly very large data quantities: experimental evidences on literature benchmarks have demonstrated the efficiency of our methods. Unfortunately, when dealing with large-scale data, there exists a gap between dimensions of benchmarks and those of real-world databases. In many industrial applications, training sets can be orders of magnitude larger than those considered in this thesis and handling them requires great engineering expertise, in memory storage or thread parallelization, for instance. Not to display any real-world application can thus be seen as a limitation of our work because we never clearly demonstrate the ability of our algorithms to tackle such problems.

This limitation exists in this thesis, as in most machine learning publications. However, even if no such real-world experiment is presented, we do not elude this issue and discuss some related aspects such as caching requirements, memory usage or training duration for all our algorithms. Besides, when used with a linear kernel, training times of SGD-QN and LaRank scale linearly with the number of examples: considering that any learning algorithms should at least pay a brief look at each training example, this is the best achievable behavior. Hence, this thesis proposes methods which could potentially fit for industrial applications.

7.1.2 Further Derivations

As much as possible, we kept the description of our innovative methods as general as possible because we always had in mind that further derivations are possible and we wanted to ease their design. Many directions can be followed to carry on with what we have been describing in this dissertation. An immediate one resides in the application of the efficient LaSVM algorithm to new

large-scale problems with the use of other refined kernel functions. For instance, [Morgado and Pereira, 2009] employs LaSVM with string-kernels for protein detection. It is also an appropriate algorithm for active learning as demonstrated in [Ertekin *et al.*, 2007a, Ertekin *et al.*, 2007b].

Similarly, the LaRank algorithm has been designed for the general topic of structured output prediction and concretely applied to two derivations, but it could be derived to problems involving trees, graphs,... or many other kind of structures. For instance, in a recent paper, [Usunier *et al.*, 2009] use LaRank to train a system designed to rank webpages.

In Chapter 3, we introduced SGD-QN for the simple setup of linear SVMs but it could be transferred on more complex problems. Even though we consider that it is not as efficient as LaSVM for learning SVMs with non-linear kernels, we believe that SGD-QN could perform very well on models with non-linear parametrization such as multi-layer perceptrons, or deep neural networks. Efficient learning of such models raises more and more attention in the literature [Hinton *et al.*, 2006, Bengio, 2009], and SGD-QN might provide an interesting alternative.

Finally, ambiguously supervised learning systems can be employed for a vast range of tasks, from speech recognition to information retrieval or image labeling. We have restricted our work to the case of semantic parsing in which we have a great interest (see Section 7.2.2). Nevertheless, the general framework described in Section 6.1 can be adapted to dozens of other applications.

7.2 AI Directions

The future research directions described in the previous section are quite straightforward because they mainly consist in direct extensions of our contributions. However there might also exist some other perspectives in which our work could apply. Remembering that machine learning is a subfield of artificial intelligence (AI) we describe two of them now.

7.2.1 Human Homology

Despite three or four decades of research on machine learning, the ability of computers to learn is still far inferior to that of humans. It can then seem natural to attempt to improve learning algorithms by imitating human behavior. Trying to mimic human learning with artificial systems might appear risky and even pretentious, but this is also an exciting challenge that can possibly afford many side benefits.

Then, if we look at the training examples that humans (or intelligent animals) employ for learning, we can gather some common properties. Indeed they (we) appear to learn from:

1. abundant data quantities,
2. continuous streams of information,
3. diverse and multimodal sources.

Following [Bengio and Le Cun, 2007], we believe human-homologous learning systems should also be trained with such data.

The combination of large-scale amounts (point 1) and data streams (point 2) tends to indicate that an online learning process is somewhat involved. But is this a strict online setup? Could an additional memory storing a fraction of training samples be appropriate? We might like to investigate if online procedures implementing the Process/Reprocess principle (introduced in Chapter 4 and 5) could share some properties with biological learning systems.

The point 3 indicates that algorithms must be able to handle diverse data formats: video, audio, text, sensors, ... Multi-view learning or transfer learning seem then appropriate. In particular, the latter aims at building systems able to to leverage knowledge previously acquired on

a given task in order to improve the efficiency and accuracy of learning in a new domain. Such methods naturally benefit from diverse sources of data. But, a framework based on ambiguous supervision could also be suitable. As we explained in Chapter 6, ambiguously supervised examples can be created within multimodal environments, by using time-synchronization for instance. An interesting challenge, in which our work could apply, could then be to conceive and study systems able to (1) automatically generate training examples out of multimodal environments and (2) train from them in an online fashion.

Some of the contributions of this dissertation could be of some interest in fields located quite far from their original purposes, because some of our work might be nicely inserted in human-inspired artificial learning systems. Of course, we do not mean that they would be useful for the models they actually train (mostly SVMs), but rather for the innovative training procedures they implement. Some models like sophisticated kernel methods, multi-layer neural-networks or reinforcement learning methods, among others, seem more likely to be ultimately learnt.

7.2.2 Natural Language Understanding

Understand, interpret or produce natural language with artificial systems have always been major challenges of AI. The complexity of the task as well as the dream of "talking to computers" have driven generation of scientists since the 70's (e.g. [Winograd *et al.*, 1972, Winston, 1976]) and the origin of natural language processing (NLP), the related subfield of AI. Besides, systems able to understand natural language would make a huge leap forward in many applicative areas. Imagine what could be done with such intelligent tools in translation, summarizing, information retrieval, speech recognition, interfacing,...

Among all concrete challenges AI can offer this one is thus our favorite. In fact, one can remark that this interest emerges and sweats now and then in this dissertation. In Section 5.3, two out of the three experimental tasks (Chunking and Part-Of-Speech tagging) are NLP related. In Chapter 6, we applied our ambiguous supervision framework to semantic parsing because this task is highly relevant to this issue.

Even if it is not directly in the scope of the thesis, we have recently started a project heading towards the ultimate goal of understanding natural language. It is destined to study and investigate ways to build artificial systems able to make the connection between language and some knowledge about their surrounding environment. This is related to both recent works [Roy and Reiter, 2005, Mooney, 2008] and old ones; the SHRLDU language by [Winograd *et al.*, 1972] for blocks worlds remaining the best existing achievement.

Hence, in Appendix C, we present a general framework and learning algorithm for the new task of *concept labeling*. This can be seen as a very basic first step to natural language understanding: one has to associate to each word of a given natural language sentence the unique physical entity (e.g. person, object, location, ...) or abstract concept it refers to. The work displayed in this appendix tends to demonstrate that grounding language using our innovative framework allows *world knowledge* and linguistic information to be used seamlessly during learning and prediction to resolve ambiguities in language.

Bibliography

[Aizerman *et al.*, 1964] M. A. Aizerman, É. M. Braverman, and L. I. Rozonoèr. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[Allen, 1995] J. Allen. *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1995.

[Altun *et al.*, 2003] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *Proceedings of the 20th International Conference on Machine Learning (ICML03)*. ACM Press, 2003.

[Amari *et al.*, 2000] S.-I. Amari, H. Park, and K. Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12:1409, 2000.

[Angelova *et al.*, 2007] A. Angelova, L. Matthies, D. M. Helmick, and P. Perona. Dimensionality reduction using automatic supervision for vision-based terrain learning. In W.Burgard, O.Brock, and C.Stachniss, editors, *Robotics: Science and Systems*, Cambridge, MA, USA, 2007. MIT Press.

[Aronszajn, 1950] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[Bakır *et al.*, 2005] G. Bakır, L. Bottou, and J. Weston. Breaking svm complexity with cross-training. In *Advances in Neural Information Processing Systems*, volume 17, pages 81–88. MIT Press, Cambridge, MA, USA, 2005.

[Bakır *et al.*, 2007] G. Bakır, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan, editors. *Predicting Structured Outputs*. MIT Press, 2007.

[Barnard and Johnson, 2005] K. Barnard and M. Johnson. Word sense disambiguation with pictures. *Artificial Intelligence*, 167(1-2):13–30, 2005.

[Becker and Le Cun, 1989] S. Becker and Y. Le Cun. Improving the convergence of back-propagation: Learning with second-order methods. In *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1989.

[Bengio and Le Cun, 2007] Y. Bengio and Y. Le Cun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, Cambridge, MA, USA, 2007.

[Bengio *et al.*, 2009] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th International Machine Learning Conference (ICML09)*. Omnipress, 2009.

[Bengio, 2009] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), 2009.

[Bennett and Bredensteiner, 2000] K.P. Bennett and E.J. Bredensteiner. Duality and geometry in SVM classifiers. In *Proceedings of the 17th International Conference on Machine Learning (ICML00)*. Morgan Kaufmann, 2000.

[Bordes and Bottou, 2005] A. Bordes and L. Bottou. The Huller: a simple and efficient online SVM. In *Machine Learning: ECML 2005*, pages 505–512. Springer Verlag, 2005. LNAI-3720.

[Bordes *et al.*, 2005] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

[Bordes *et al.*, 2007] A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *Proceedings of the 24th International Machine Learning Conference (ICML07)*. OmniPress, 2007.

[Bordes *et al.*, 2008] A. Bordes, N. Usunier, and L. Bottou. Sequence labelling SVMs trained in one pass. In *ECML PKDD 2008*, pages 146–161. Springer, 2008.

[Bordes *et al.*, 2009] A. Bordes, L. Bottou, and P. Gallinari. SGD-QN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

[Bottou and Bousquet, 2008] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20. MIT Press, Cambridge, MA, 2008.

[Bottou and Le Cun, 2005] L. Bottou and Y. Le Cun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005.

[Bottou and Lin, 2007] L. Bottou and C.-J. Lin. Support vector machine solvers. In *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

[Bottou, 1998] L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[Bottou, 2007] L. Bottou. Stochastic gradient descent on toy problems, 2007. `http://leon.bottou.org/projects/sgd`.

[Boyd and Vandenberghe, 2004] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[Campbell *et al.*, 2000] C. Campbell, N. Cristianini, and A. J. Smola. Query learning with large margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning (ICML00)*. Morgan Kaufmann, 2000.

[Cauwenberghs and Poggio, 2001] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Processing Systems*, volume 13. MIT Press, 2001.

[Cesa-Bianchi and Lugosi, 2006] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[Cesa-Bianchi *et al.*, 2004] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[Cesa-Bianchi *et al.*, 2005] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Worst-case analysis of selective sampling for linear-threshold algorithms. In *Advances in Neural Information Processing Systems*, volume 17, pages 241–248. MIT Press, 2005.

[Chang and Lin, 2001 2004] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. Technical report, Computer Science and Information Engineering, National Taiwan University, 2001-2004.

[Chen and Mooney, 2008] D.L. Chen and R.J. Mooney. Learning to sportscast: A test of grounded language acquisition. In *Proceedings of the 25th International Machine Learning Conference (ICML08)*. OmniPress, 2008.

[Cohn *et al.*, 1990] D. Cohn, L. Atlas, and R. Ladner. Training connectionist networks with queries and selective sampling. In *Advances in Neural Information Processing Systems*, volume 2. Morgan Kaufmann, San Francisco, CA, USA, 1990.

[Collins and Roark, 2004] M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL04)*. Association for Computational Linguistics, 2004.

[Collins *et al.*, 2008] M. Collins, A. Globerson, T. Koo, X.Carreras, and P. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *Journal of Machine Learning Research*, 9:1775–1822, 2008.

[Collins, 2002] M. Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL Workshop on Empirical methods in natural language processing (EMNLP02)*. Association for Computational Linguistics, 2002.

[Collobert and Bengio, 2001] R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[Collobert and Weston, 2008] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Machine Learning Conference (ICML08)*. OmniPress, 2008.

[Collobert *et al.*, 2002] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002.

[Cortes and Vapnik, 1995] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[Cour *et al.*, 2008] T. Cour, C. Jordan, E. Miltsakaki, and B. Taskar. Movie/Script: Alignment and Parsing of Video and Text Transcription. In *Proceedings of the 10th European Conference on Computer Vision (ECCV08)*. Springer-Verlag, 2008.

[Cour *et al.*, 2009] T. Cour, B. Sapp, C. Jordan, and B. Taskar. Learning from ambiguously labeled images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition'09 (CVPR09)*, 2009.

[Crammer and Singer, 2001] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[Crammer and Singer, 2003] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

[Crammer and Singer, 2005] K. Crammer and Y. Singer. Loss Bounds for Online Category Ranking. In *Proceedings of the 18th Annual Conference on Computational Learning Theory (COLT05)*, 2005.

[Crammer *et al.*, 2004] K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, 2004.

[Crammer *et al.*, 2006] K. Crammer, O. Dekel, J. Keshet, Y. Singer, and M. K. Warmuth. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[Crisp and Burges, 2000] D.J. Crisp and C.J.C. Burges. A geometric interpretation of ν-SVM classifiers. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[Cristianini and Shawe-Taylor, 2000] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[Daumé III and Marcu, 2005] H. Daumé III and D. Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd International Conference on Machine Learning (ICML05)*, 2005.

[Daumé III *et al.*, 2005] H. Daumé III, J. Langford, and D. Marcu. Search-based structured prediction as classification. In *NIPS*Workshop on Advances in Structured Learning for Text and Speech Processing*, 2005.

[Denoyer and Gallinari, 2006] L. Denoyer and P. Gallinari. The XML document mining challenge. In *Advances in XML Information Retrieval and Evaluation, 5th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX06)*, volume 3977 of *Lecture Notes in Computer Science*, 2006.

[Domingo and Watanabe, 2000] C. Domingo and O. Watanabe. MadaBoost: a modification of AdaBoost. In *Proceedings of the 13th Annual Conference on Computational Learning Theory (COLT00)*, 2000.

[Driancourt, 1994] X. Driancourt. *Optimisation par descente de gradient stochastique de systèmes modulaires combinant réseaux de neurones et programmation dynamique*. PhD thesis, Université Paris XI, Orsay, France, 1994.

[Eisenberg and Rivest, 1990] B. Eisenberg and R. Rivest. On the sample complexity of PAC learning using random and chosen examples. In *Proceedings of the 3rd Annual ACM Workshop on Computational Learning Theory*. Morgan Kaufmann, 1990.

[Ertekin *et al.*, 2007a] S. Ertekin, J. Huang, L. Bottou, and L. C. Giles. Learning on the border: active learning in imbalanced data classification. In *Proceedings of the 16th ACM conference on information and knowledge management (CIKM07)*. ACM Press, 2007.

[Ertekin *et al.*, 2007b] S. Ertekin, J. Huang, and L. C. Giles. Active learning for class imbalance problem. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR07)*. ACM Press, 2007.

[Fabian, 1973] V. Fabian. Asymptotically efficient stochastic approximation: The rm case. *Annals of Statistics*, 1(3):486–495, 1973.

[Fan *et al.*, 2008] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[Fedorov, 1972] V. V. Fedorov. *Theory of Optimal Experiments*. Academic Press, New York, 1972.

[Feldman *et al.*, 1996] J. Feldman, G. Lakoff, D. Bailey, S. Narayanan, T. Regier, and A. Stolcke. L0 the first five years of an automated language acquisition project. *Artificial Intelligence Review*, 10(1):103–129, 1996.

[Fleischman and Roy, 2005] M. Fleischman and D. Roy. Intentional context in situated language learning. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNNL05)*, 2005.

[Fleischman and Roy, 2007] M. Fleischman and D. Roy. Situated Models of Meaning for Sports Video Retrieval. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL07)*, 2007.

[Franc and Sonnenburg, 2008] V. Franc and S. Sonnenburg. Ocas optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th International Machine Learning Conference (ICML08)*. Omnipress, 2008.

[Freund and Schapire, 1998] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the 15th International Conference on Machine Learning (ICML98)*. Morgan Kaufmann, 1998.

[Frieß *et al.*, 1998] T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel Adatron algorithm: a fast and simple learning procedure for support vector machines. In *Proceedings of the 15th International Conference on Machine Learning (ICML98)*. Morgan Kaufmann, 1998.

[Furey *et al.*, 2000] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, October 2000.

[Gentile, 2001] C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

[Gilbert, 1966] E.G. Gilbert. Minimizing the quadratic form on a convex set. *SIAM Journal of Control*, 4:61–79, 1966.

[Graepel *et al.*, 2000] T. Graepel, R. Herbrich, and R. C. Williamson. From margin to sparsity. In *Advances in Neural Information Processing Systems*, volume 13, pages 210–216. MIT Press, 2000.

[Graf *et al.*, 2005] H.-P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik. Parallel support vector machines: The Cascade SVM. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005.

[Gramacy *et al.*, 2003] R. Gramacy, M. Warmuth, S. Brandt, and I. Ari. Adaptive caching by refetching. In *Advances in Neural Information Processing Systems*, volume 15, pages 1465–1472. MIT Press, 2003.

[Guyon *et al.*, 1993] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, 1993.

[Haffner, 2002] P. Haffner. Escaping the convex hull with extrapolated vector machines. In *Advances in Neural Information Processing Systems*, volume 14, pages 753–760. MIT Press, 2002.

[Har-Peled *et al.*, 2002] S. Har-Peled, D. Roth, and D. Zimak. Constraint classification for multiclass classification and ranking. In *Advances in Neural Information Processing Systems*, volume 13, pages 785–792. MIT Press, 2002.

[Harnad, 1990] S. Harnad. The symbol grounding problem. *Physica D*, 42(1-3):335–346, 1990.

[Hildreth, 1957] C. Hildreth. A quadratic programming procedure. *Naval Research Logistics Quarterly*, 4:79–85, 1957. Erratum, ibid. p361.

[Hinton *et al.*, 2006] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[Hsieh *et al.*, 2008] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Machine Learning Conference (ICML08)*. Omnipress, 2008.

[Hsu and Lin, 2002] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[Joachims, 1999] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods – Support Vector Learning*, pages 169–184. MIT Press, 1999.

[Joachims, 2000] T. Joachims. *The Maximum-Margin Approach to Learning Text Classifiers: Methods, Theory, and Algorithms*. PhD thesis, Universität Dortmund, Informatik, LS VIII, 2000.

[Joachims, 2006] T. Joachims. Training linear svms in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD06)*. ACM Press, 2006.

[Kassel, 1995] R Kassel. *A Comparison of Approaches on Online Handwritten Character Recognition*. PhD thesis, MIT. Spoken Language System Group, 1995.

[Kate and Mooney, 2006] R.J. Kate and R. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL06)*, volume 45, 2006.

[Kate and Mooney, 2007] R.J. Kate and R.J. Mooney. Learning language semantics from ambiguous supervision. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI07)*, volume 22, 2007.

[Keerthi and Gilbert, 2002] S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46:351–360, 2002.

[Keerthi *et al.*, 1999] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical report, TR-ISL-99-03, Indian Institute of Science, Bangalore, 1999.

[Kingsbury and Palmer, 2002] P. Kingsbury and M. Palmer. From treebank to propbank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, 2002.

[Kudoh and Matsumoto, 2000] T. Kudoh and Y. Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of the 4th Conference on Computational Natural Language Learning (CoNLL00)*, 2000.

[Lafferty *et al.*, 2001] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML01)*, 2001.

[Laskov *et al.*, 2004] P. Laskov, C. Schäfer, and I. Kotenko. Intrusion detection in unlabeled data with quarter-sphere support vector machines. In *Proceedings of Conference on Detection of Intrusions, Malware and Vulnerability Assessment (DIMVA'04)*, 2004.

[Laskov *et al.*, 2006] P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller. Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, 2006.

[Le Cun *et al.*, 1997] Y. Le Cun, L. Bottou, and Y. Bengio. Reading checks with graph transformer networks. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 151–154, Munich, 1997. IEEE.

[Le Cun *et al.*, 1998] Y. Le Cun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[Le Cun *et al.*, 2007] Y. Le Cun, S. Chopra, R. Hadsell, Huang F.-J., and M. Ranzato. A tutorial on energy-based learning. In Bakır et al. [2007], pages 192–241.

[Lewis *et al.*, 2004] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[Li and Long, 2002] Y. Li and P. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46:361–387, 2002.

[Lin, 2001] C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

[Littlestone and Warmuth, 1986] N. Littlestone and M. Warmuth. Relating data compression and learnability. Technical report, Technical Report University of California Santa Cruz, 1986.

[Loosli *et al.*, 2007] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

[Ma *et al.*, 2009] J. Ma, L. K. Saul, S. Savage, and G. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th International Machine Learning Conference (ICML09)*. OmniPress, 2009.

[MacKay, 1992] D. J. C. MacKay. Information based objective functions for active data selection. *Neural Computation*, 4:589–603, 1992.

[Maes *et al.*, 2007] F. Maes, L. Denoyer, and P. Gallinari. Sequence labelling with reinforcement learning and ranking algorithms. In *Machine Learning: ECML 2007*, Warsaw, Poland, 2007.

[Manning, 1999] C. Manning. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[Miller, 1995] G.A. Miller. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Mooney, 2008] R. Mooney. Learning to connect language and perception. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI08)*, 2008.

[Morgado and Pereira, 2009] L. Morgado and C. Pereira. Incremental kernel machines for protein remote homology detection. In *Hybrid Artificial Intelligence Systems*, Lecture Notes in Computer Science, pages 409–416. Springer, 2009.

[Murata and Amari, 1999] N. Murata and S.-I. Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1):3–28, 1999.

[Murata and Onoda, 2002] H. Murata and T. Onoda. Estimation of power consumption for household electric appliances. In *Advances in Neural Information Processing Systems*, volume 13, pages 2299–2303. MIT Press, 2002.

[Nilsson, 1965] N. J. Nilsson. *Learning machines: Foundations of Trainable Pattern Classifying Systems*. McGraw–Hill, 1965.

[Nocedal, 1980] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[Novikoff, 1962] A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12. Polytechnic Institute of Brooklyn, 1962.

[Platt, 1999] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, pages 185–208. MIT Press, 1999.

[Pradhan *et al.*, 2004] S. S. Pradhan, W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky. Shallow semantic parsing using support vector machines. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL04)*, 2004.

[Rabiner and Juang, 1986] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 3(1), January 1986.

[Rifkin and Klautau, 2004] R. M. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[Rosenblatt, 1958] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. In *Psychological Review*, volume 65, pages 386–408, 1958.

[Roy and Reiter, 2005] D. Roy and E. Reiter. Connecting language to the world. *Artificial Intelligence*, 167(1-2):1–12, September 2005.

[Russell *et al.*, 1995] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*. Prentice Hall Englewood Cliffs, NJ, 1995.

[Schohn and Cohn, 2000] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning (ICML00)*. Morgan Kaufmann, 2000.

[Schölkopf and Smola, 2002] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[Schraudolph *et al.*, 2007] N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AIstats07)*. Society for Artificial Intelligence and Statistics, 2007.

[Schraudolph, 1999] N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN99)*, 1999.

[Schrijver, 1986] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1986.

[Sha and Pereira, 2003] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL03)*. Association for Computational Linguistics, 2003.

[Shalev-Shwartz and Singer, 2007a] S. Shalev-Shwartz and Y. Singer. A primal-dual perspective of online learning algorithms. *Machine Learning*, 69(2-3):115–142, 2007.

[Shalev-Shwartz and Singer, 2007b] S. Shalev-Shwartz and Y. Singer. A unified algorithmic approach for efficient online label ranking. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AIstats07)*. Society for Artificial Intelligence and Statistics, 2007.

[Shalev-Shwartz *et al.*, 2007] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning (ICML07)*. OmniPress, 2007.

[Siskind, 1994] J.M. Siskind. Grounding language in perception. *Artificial Intelligence Review*, 8(5):371–391, 1994.

[Smola *et al.*, 2008] A. Smola, S.V.N. Vishwanathan, and Q. Le. Bundle methods for machine learning. In *Advances in Neural Information Processing Systems*, volume 20, pages 1377–1384. MIT Press, Cambridge, MA, 2008.

[Sonnenburg *et al.*, 2008] S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag. Pascal large scale learning challenge. ICML'08 Workshop, 2008. `http://largescale.first.fraunhofer.de`.

[Soon *et al.*, 2001] W.M. Soon, H.T. Ng, and D.C.Y. Lim. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544, 2001.

[Steinwart, 2004] I. Steinwart. Sparseness of support vector machines – some asymptotically sharp bounds. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004.

[Takahashi and Nishi, 2003] N. Takahashi and T. Nishi. On termination of the SMO algorithm for support vector machines. In *Proceedings of International Symposium on Information Science and Electrical Engineering 2003*, 2003.

[Taskar *et al.*, 2004] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004.

[Taskar *et al.*, 2005] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: a large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning (ICML05)*. ACM Press, 2005.

[Taskar, 2004] B. Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.

[Thibadeau, 1986] R. Thibadeau. Artificial perception of actions. *Cognitive Science*, 10(2):117–149, 1986.

[Tong and Koller, 2000] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proceedings of the 17th International Conference on Machine Learning (ICML00)*. Morgan Kaufmann, 2000.

[Toutanova *et al.*, 2003] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (HLT-NAACL03)*. Association for Computational Linguistics, 2003.

[Tsang *et al.*, 2005] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Very large SVM training using core vector machines. In *Proceedings of the 10th International Conference on Artificial Intelligence and Statistics (AIstats05)*. Society for Artificial Intelligence and Statistics, 2005.

[Tsochantaridis *et al.*, 2005] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[Usunier *et al.*, 2009] N. Usunier, D. Buffoni, and P. Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th International Machine Learning Conference (ICML09)*. Omnipress, 2009.

[Vapnik and Lerner, 1963] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[Vapnik *et al.*, 1984] V. N. Vapnik, T. G. Glaskova, V. A. Koscheev, A. I. Mikhailski, and A. Y. Chervonenkis. *Algorihms and Programs for Dependency Estimation*. Nauka, 1984.

[Vapnik, 1982] V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[Vapnik, 1998] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[Vishwanathan *et al.*, 2003] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM. In *Proceedings of the 20th International Conference on Machine Learning (ICML03)*. ACM Press, 2003.

[Von Ahn *et al.*, 2008] L. Von Ahn, B. Maurer, C. Mcmillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, August 2008.

[Von Ahn, 2006] L. Von Ahn. Games with a purpose. *IEEE Computer Magazine*, pages 96–98, June 2006.

[Warmuth *et al.*, 2003] M. K. Warmuth, J. Liao, G. Ratsch, M. Mathieson, S. Putta, and C. Lemmen. Active learning with support vector machines in the drug discovery process. *Journal of Chemical Information and Computer Sciences*, 43(2):667–673, 2003.

[Weston and Watkins, 1998] J. Weston and C. Watkins. Multi-class support vector machines. Technical report, Technical Report Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998.

[Weston *et al.*, 2005] J. Weston, A. Bordes, and L. Bottou. Online (and offline) on an even tighter budget. In *Proceedings of the 10th International Conference on Artificial Intelligence and Statistics (AIstats05)*. Society for Artificial Intelligence and Statistics, 2005.

[Winograd *et al.*, 1972] T. Winograd, M.G. Barbour, and C.R. Stocking. *Understanding natural language*. Academic Press New York, 1972.

[Winston, 1976] P.H. Winston. The psychology of computer vision. *Pattern Recognition*, 8(3):193–193, 1976.

[Wong and Mooney, 2007] Y.W. Wong and R. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL07)*, volume 45, 2007.

[Yu and Ballard, 2004] C. Yu and D.H. Ballard. On the Integration of Grounding Language and Learning Objects. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence (AAAI04)*, 2004.

[Zettlemoyer and Collins, 2005] L.S. Zettlemoyer and M. Collins. Learning to Map sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of Uncertainty in Artificial Intelligence (UAI05)*, 2005.

[Zhang *et al.*, 2002] T. Zhang, F. Damerau, and D. Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2:615–637, 2002.

[Zoutendijk, 1960] G. Zoutendijk. *Methods of Feasible Directions*. Elsevier, 1960.

A

Personal Bibliography

The work described in this dissertation has been the object of several awards, publications and talks. We summarize them here.

Awards

Winner of the *PASCAL Large Scale Learning Challenge "Wild Track"* (2008).
SGD-QN algorithm ranked 1^{st} *ex-eaquo* over 42 international competitors.
Challenge organized by S. Sonnenburg, V. Franc, E. Yom-Tov and M. Sebag.
`http://largescale.first.fraunhofer.de/`

Best Student Paper Award at ICML (2007).
For the paper *Solving MultiClass Support Vector Machines with LaRank*.

Journal Publications

SGDQN: Careful Quasi-Newton Stochastic Gradient Descent (2009)
Antoine Bordes, Léon Bottou and Patrick Gallinari. in Journal of Machine Learning Research. 10:1737-1754. MIT Press.

Fast Kernel Classifiers with Online and Active Learning (2005).
Antoine Bordes, Seyda Ertekin, Jason Weston and Léon Bottou. in Journal of Machine Learning Research, 6:1579-1619. MIT Press.

Conference Proceedings

Sequence Labeling with SVMs Trained in One Pass (2008)
Antoine Bordes, Nicolas Usunier and Léon Bottou. in ECML PKDD 2008, Part I, 146-161, Springer Verlag.

Solving MultiClass Support Vector Machines with LaRank (2007).
Antoine Bordes, Léon Bottou, Patrick Gallinari and Jason Weston. in Proceedings of the 24th International Machine Learning Conference (ICML07). OmniPress.

The Huller: a Simple and Efficient Online SVM (2005).
Antoine Bordes and Léon Bottou. in Machine Learning: ECML 2005, 505-512. Springer Verlag.

Online (and Offline) Learning on an Even Tighter Budget (2005).
Jason Weston, Antoine Bordes and Léon Bottou. in Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTAT05), 413-420. Society for Artificial Intelligence and Statistics.

Selected Talks

Towards Understanding Situated Text: Concept Labeling & Extensions (2009).
Antoine Bordes, Nicolas Usunier, Ronan Collobert and Jason Weston. at the Learning Workshop, Clearwater, USA. 13-17 April 2009.

SGDQN, LaRank: Fast Optimizers for Linear SVMs (2008).
Antoine Bordes and Léon Bottou. at ICML*2008 Workshop for PASCAL Large Scale Learning Challenge, Helsinki, Finland. 9 July 2008.

Learning To Label Sequences in One Pass (2008).
Antoine Bordes, Nicolas Usunier and Léon Bottou. at the Learning Workshop, Snowbird, USA. 1-4 April 2008.

Large-Scale Sequence Labeling (2007).
Antoine Bordes and Léon Bottou. at NIPS*2007 Workshop on Efficient Machine Learning, Whistler, Canada. 7-8 December 2007.

B

Convex Programming with Witness Families

This appendix presents theoretical elements about convex programming algorithms that rely on successive direction searches. Results are presented for the case where directions are selected from a well chosen finite pool, like SMO [Platt, 1999], and for the stochastic algorithms, like the online and active SVM discussed in Chapter 4.

Consider a compact convex subset $\mathcal{F}$ of $\mathcal{R}^n$ and a concave function f defined on $\mathcal{F}$. We assume that f is twice differentiable with continuous derivatives. This appendix discusses the maximization of function f over set F.

$$\max_{x \in \mathcal{F}} f(x) \tag{B.1}$$

This discussion starts with some results about feasible directions. Then it introduces the notion of witness family of directions which leads to a more compact characterization of the optimum. Finally it presents maximization algorithms and establishes their convergence to approximate solutions

B.1 Feasible Directions

Notations Given a point $x \in \mathcal{F}$ and a direction $\boldsymbol{u} \in \mathcal{R}^n_* = \mathcal{R}^n$, let

$$\begin{aligned} \phi(x, \boldsymbol{u}) &= \max\{\lambda \geq 0 \mid x + \lambda \boldsymbol{u} \in \mathcal{F}\} \\ f^*(x, \boldsymbol{u}) &= \max\{f(x + \lambda \boldsymbol{u}),\, x + \lambda \boldsymbol{u} \in \mathcal{F},\, \lambda \geq 0\} \end{aligned}$$

In particular we write $\phi(x, \mathbf{0}) = \infty$ and $f^*(x, \mathbf{0}) = f(x)$.

Definition 1 *The cone of feasible directions in $x \in \mathcal{F}$ is the set*

$$\mathcal{D}_x = \{\boldsymbol{u} \in \mathcal{R}^n \mid \phi(x, \boldsymbol{u}) > 0\}$$

All the points $x + \lambda \boldsymbol{u}$, $0 \leq \lambda \leq \phi(x, \boldsymbol{u})$ belong to $\mathcal{F}$ because $\mathcal{F}$ is convex. Intuitively, a direction $\boldsymbol{u} \neq 0$ is feasible in x when we can start from x and make a little movement along direction $\boldsymbol{u}$ without leaving the convex set $\mathcal{F}$.

Proposition 14 *Given $x \in \mathcal{F}$ and $\boldsymbol{u} \in \mathcal{R}^n$,*

$$f^*(x, \boldsymbol{u}) > f(x) \quad \Longleftrightarrow \quad \left\{ \begin{array}{l} \boldsymbol{u}' \boldsymbol{\nabla} f(x) > 0 \\ \boldsymbol{u} \in \mathcal{D}_x \end{array} \right.$$

Proof Assume $f^*(x,\boldsymbol{u}) > f(x)$. Direction $\boldsymbol{u} \neq \mathbf{0}$ is feasible because the maximum $f^*(x,\boldsymbol{u})$ is reached for some $0 < \lambda^* \leq \phi(x,\boldsymbol{u})$. Let $\nu \in [0,1]$. Since set $\mathcal{F}$ is convex, $x + \nu\lambda^*\boldsymbol{u} \in \mathcal{F}$. Since function f is concave, $f(x + \nu\lambda^*\boldsymbol{u})) \geq (1-\nu)f(x) + \nu f^*(x,\boldsymbol{u})$. Writing a first order expansion when $\nu \to 0$ yields $\lambda^*\boldsymbol{u}'\boldsymbol{\nabla}f(x) \geq f^*(x,\boldsymbol{u}) - f(x) > 0$. Conversely, assume $\boldsymbol{u}'\boldsymbol{\nabla}f(x) > 0$ and $\boldsymbol{u} \neq 0$ is a feasible direction. Recall $f(x+\lambda\boldsymbol{u}) = f(x) + \lambda\boldsymbol{u}'\boldsymbol{\nabla}f(x) + o(\lambda)$. Therefore we can choose $0 < \lambda_0 \leq \phi(x,\boldsymbol{u})$ such that $f(x+\lambda_0\boldsymbol{u}) > f(x) + \lambda_0\boldsymbol{u}'\boldsymbol{\nabla}f(x)/2$. Therefore $f^*(x,\boldsymbol{u}) \geq f(x+\lambda_0\boldsymbol{u}) > f(x)$. □

Theorem 15 ([Zoutendijk, 1960] page 22) *The following assertions are equivalent:*

i) x *is a solution of problem (B.1).*

ii) $\forall \boldsymbol{u} \in \mathcal{R}^n \quad f^*(x,\boldsymbol{u}) \leq f(x)$.

iii) $\forall \boldsymbol{u} \in \mathcal{D}_x \quad \boldsymbol{u}'\boldsymbol{\nabla}f(x) \leq 0$.

Proof The equivalence between assertions (*ii*) and (*iii*) results from Proposition 14. Assume assertion (*i*) is true. Assertion (*ii*) is necessarily true because $f^*(\boldsymbol{u},x) \leq \max_{\mathcal{F}} f = f(x)$. Conversely, assume assertion (*i*) is false. Then there is $y \in \mathcal{F}$ such that $f(y) > f(x)$. Therefore $f^*(x, y-x) > f(x)$ and assertion (*ii*) is false. □

B.2 Witness Families

We now seek to improve this theorem. Instead of considering all feasible directions in $\mathcal{R}^n$, we wish to only consider the feasible directions from a smaller set $\mathcal{U}$.

Proposition 16 *Let $x \in \mathcal{F}$ and $\mathbf{v}_1 \ldots \mathbf{v}_k \in \mathcal{D}_x$ be feasible directions. Every positive linear combination of $\mathbf{v}_1 \ldots \mathbf{v}_k$ (i.e. a linear combination with positive coefficients) is a feasible direction.*

Proof Let $\boldsymbol{u}$ be a positive linear combination of the $\mathbf{v}_i$. Since the $\mathbf{v}_i$ are feasible directions there are $y_i = x + \lambda_i\mathbf{v}_i \in \mathcal{F}$, and $\boldsymbol{u}$ can be written as $\sum_i \gamma_i(y_i - x)$ with $\gamma_i \geq 0$. Direction $\boldsymbol{u}$ is feasible because the convex $\mathcal{F}$ contains $(\sum \gamma_i y_i) \,/\, (\sum \gamma_i) = x + (1/\sum \gamma_i)\,\boldsymbol{u}$. □

Definition 2 *A set of directions $\mathcal{U} \subset \mathcal{R}^n_*$ is a "witness family for $\mathcal{F}$" when, for any point $x \in \mathcal{F}$, any feasible direction $\boldsymbol{u} \in \mathcal{D}_x$ can be expressed as a positive linear combination of a finite number of feasible directions $\mathbf{v}_j \in \mathcal{U} \cap \mathcal{D}_x$.*

This definition directly leads to an improved characterization of the optima.

Theorem 17 *Let $\mathcal{U}$ be a witness family for convex set $\mathcal{F}$.*
The following assertions are equivalent:

i) x *is a solution of problem* (B.1).

ii) $\forall \boldsymbol{u} \in \mathcal{U} \quad f^*(x,\boldsymbol{u}) \leq f(x)$.

iii) $\forall \boldsymbol{u} \in \mathcal{U} \cap \mathcal{D}_x \quad \boldsymbol{u}'\boldsymbol{\nabla}f(x) \leq 0$.

Proof The equivalence between assertions (*ii*) and (*iii*) results from Proposition 14. Assume assertion (*i*) is true. Theorem 15 implies that assertion (*ii*) is true as well. Conversely, assume assertion (*i*) is false. Theorem 15 implies that there is a feasible direction $\boldsymbol{u} \in \mathcal{R}^n$ on point x such that $\boldsymbol{u}'\boldsymbol{\nabla}f(x) > 0$. Since $\mathcal{U}$ is a witness family, there are positive coefficients $\gamma_1 \ldots \gamma_k$ and feasible directions $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathcal{U} \cap \mathcal{D}_x$ such that $\boldsymbol{u} = \sum \gamma_i \mathbf{v}_i$. We have then $\sum \gamma_j \mathbf{v}_j'\boldsymbol{\nabla}f(x) > 0$. Since all coefficients γ_j are positive, there is at least one term j_0 such that $\mathbf{v}_{j_0}'\boldsymbol{\nabla}f(x) > 0$. Assertion (*iii*) is therefore false. □

The following proposition provides an example of witness family for the convex domain $\mathcal{F}_{\rm s}$ that appears in the SVM QP problem (2.9).

Proposition 18 *Let $(\mathbf{e}_1 \dots \mathbf{e}_n)$ be the canonical basis of $\mathcal{R}^n$. Set $\mathcal{U}_{\rm s} = \{\mathbf{e}_i - \mathbf{e}_j, i \neq j\}$ is a witness family for convex set $\mathcal{F}_{\rm s}$ defined by the constraints*

$$x \in \mathcal{F}_{\rm s} \quad \Longleftrightarrow \quad \left\{ \begin{array}{l} \forall\, i \quad A_i \le x_i \le B_i \\ \sum_i x_i = 0 \end{array} \right.$$

Proof Let $\boldsymbol{u} \in \mathcal{R}^n_*$ be a feasible direction in $x \in \mathcal{F}_{\rm s}$. Since $\boldsymbol{u}$ is a feasible direction, there is $\lambda > 0$ such that $y = x + \lambda \boldsymbol{u} \in \mathcal{F}_{\rm s}$. Consider the subset $\mathcal{B} \subset \mathcal{F}_{\rm s}$ defined by the constraints:

$$\boldsymbol{z} \in \mathcal{B} \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \forall\, i,\ A_i \le \min(x_i, y_i) \le z_i \le \max(x_i, y_i) \le B_i \\ \sum_i z_i = 0 \end{array} \right.$$

Let us recursively define a sequence of points $\boldsymbol{z}(j) \in \mathcal{B}$. We start with $\boldsymbol{z}(0) = x \in \mathcal{B}$. For each $t \ge 0$, we define two sets of coordinate indices $I_t^+ = \{i \,|\, z_i(t) < y_i\}$ and $I_t^- = \{j \,|\, z_j(t) > y_j\}$. The recursion stops if either set is empty. Otherwise, we choose $i \in I_t^+$ and $j \in I_t^-$ and define $\boldsymbol{z}(t+1) = \boldsymbol{z}(t) + \gamma(t)\, \mathbf{v}(t) \in \mathcal{B}$ with $\mathbf{v}(t) = \mathbf{e}_i - \mathbf{e}_j \in \mathcal{U}_{\rm s}$ and $\gamma(t) = \min(y_i - z_i(t), z_j(t) - y_j) > 0$. Intuitively, we move towards y along direction $\mathbf{v}(t)$ until we hit the boundaries of set $\mathcal{B}$.

Each iteration removes at least one of the indices i or j from sets I_t^+ and I_t^-. Eventually one of these sets gets empty and the recursion stops after a finite number k of iterations. The other set is also empty because

$$\sum_{i \in I_k^+} |y_i - z_i(k)| - \sum_{i \in I_k^-} |y_i - z_i(k)| \ = \ \sum_{i=1}^n y_i - z_i(k) \ = \ \sum_{i=1}^n y_i - \sum_{i=1}^n z_i(k) \ = \ 0.$$

Therefore $\boldsymbol{z}(k) = y$ and $\lambda \boldsymbol{u} = y - x = \sum_t \gamma(t)\, \mathbf{v}(t)$. Moreover the $\mathbf{v}(t)$ are feasible directions on x because $\mathbf{v}(t) = \mathbf{e}_i - \mathbf{e}_j$ with $i \in I_t^+ \subset I_0^+$ and $j \in I_t^- \subset I_0^-$. □

Assertion *(iii)* in Theorem 17 then yields the following necessary and sufficient optimality criterion for the SVM QP problem (2.9).

$$\forall\, (i,j) \in \{1 \dots n\}^2 \qquad x_i < B_i \text{ and } x_j > A_j \ \Rightarrow \ \frac{\partial f}{\partial x_i}(x) - \frac{\partial f}{\partial x_j}(x) \le 0$$

Different constraint sets call for different choices of witness family. For instance, it is sometimes useful to disregard the equality constraint in the SVM polytope $\mathcal{F}_{\rm s}$. Along the lines of Proposition 18, it is quite easy to prove that $\{\pm \mathbf{e}_i,\, i = 1 \dots n\}$ is a witness family. Theorem 17 then yields an adequate optimality criterion.

B.3 Finite Witness Families

This subsubsection deals with *finite witness families*. Theorem 20 shows that $\mathcal{F}$ is necessarily a convex polytope, that is a bounded set defined by a finite number of linear of linear equality and inequality constraints [Schrijver, 1986].

Proposition 19 *Let $\mathcal{C}_x = \{x + \boldsymbol{u}\ ,\ \boldsymbol{u} \in \mathcal{D}_x\}$ for $x \in \mathcal{F}$. Then $\mathcal{F} = \bigcap_{x \in \mathcal{F}} \mathcal{C}_x$.*

Proof We first show that $\mathcal{F} \subset \bigcap_{x \in \mathcal{F}} \mathcal{C}_x$. Indeed $\mathcal{F} \subset \mathcal{C}_x$ for all x because every point $\boldsymbol{z} \in \mathcal{F}$ defines a feasible direction $\boldsymbol{z} - x \in \mathcal{D}_x$.

Conversely, Let $\boldsymbol{z} \in \bigcap_{x \in \mathcal{F}} \mathcal{C}_x$ and assume that $\boldsymbol{z}$ does not belong to $\mathcal{F}$. Let $\hat{\boldsymbol{z}}$ be the projection of $\boldsymbol{z}$ on $\mathcal{F}$. We know that $\boldsymbol{z} \in \mathcal{C}_{\hat{\boldsymbol{z}}}$ because $\boldsymbol{z} \in \bigcap_{x \in \mathcal{F}} \mathcal{C}_x$. Therefore $\boldsymbol{z} - \hat{\boldsymbol{z}}$ is a feasible direction in $\hat{\boldsymbol{z}}$. Choose $0 < \lambda < \phi\ \ (\hat{\boldsymbol{z}}, \boldsymbol{z} - \hat{\boldsymbol{z}})$. We know that $\lambda < 1$ because $\boldsymbol{z}$ does not belong to $\mathcal{F}$. But then $\hat{\boldsymbol{z}} + \lambda(\boldsymbol{z} - \hat{\boldsymbol{z}}) \in \mathcal{F}$ is closer to $\boldsymbol{z}$ than $\hat{\boldsymbol{z}}$. This contradicts the definition of the projection $\hat{\boldsymbol{z}}$. □

Theorem 20 *Let $\mathcal{F}$ be a bounded convex set.*
If there is a finite witness family for $\mathcal{F}$, then $\mathcal{F}$ is a convex polytope[1].

Proof Consider a point $x \in \mathcal{F}$ and let $\{\mathbf{v}_1 \dots \mathbf{v}_k\} = \mathcal{U} \cap \mathcal{D}_x$. Proposition 16 and Definition 2 imply that $\mathcal{D}_x$ is the polyhedral cone $\{\boldsymbol{z} = \sum \gamma_i \mathbf{v}_i,\ \gamma_i \geq 0\}$ and can be represented [Schrijver, 1986] by a finite number of linear equality and inequality constraints of the form $\boldsymbol{n}\boldsymbol{z} \leq 0$ where the directions $\boldsymbol{n}$ are unit vectors. Let $\mathcal{K}_x$ be the set of these unit vectors. Equality constraints arise when the set $\mathcal{K}_x$ contains both $\boldsymbol{n}$ and $-\boldsymbol{n}$. Each set $\mathcal{K}_\S$ depends only on the subset $\{\mathbf{v}_1 \dots \mathbf{v}_k\} = \mathcal{U} \cap \mathcal{D}_x$ of feasible witness directions in x. Since the finite set $\mathcal{U}$ contains only a finite number of potential subsets, there is only a finite number of distinct sets $\mathcal{K}_x$.

Each set $\mathcal{C}_x$ is therefore represented by the constraints $\boldsymbol{n}\,\boldsymbol{z} \leq \boldsymbol{n}\,x$ for $\boldsymbol{n} \in \mathcal{K}_x$. The intersubsubsection $\mathcal{F} = \bigcap_{x \in \mathcal{F}} \mathcal{C}_x$ is then defined by all the constraints associated with $\mathcal{C}_x$ for any $x \in \mathcal{F}$. These constraints involve only a finite number of unit vectors $\boldsymbol{n}$ because there is only a finite number of distinct sets $\mathcal{K}_x$.

Inequalities defined by the same unit vector $\boldsymbol{n}$ can be summarized by considering only the most restrictive right hand side. Therefore $\mathcal{F}$ is described by a finite number of equality and inequality constraints. Since $\mathcal{F}$ is bounded, it is a polytope. □

A convex polytope comes with useful continuity properties.

Proposition 21 *Let $\mathcal{F}$ be a polytope, and let $\boldsymbol{u} \in \mathcal{R}^n$ be fixed.*
Functions $x \mapsto \phi(x, \boldsymbol{u})$ and $x \mapsto f^*(x, \boldsymbol{u})$ are uniformly continous on $\mathcal{F}$.

Proof The polytope $\mathcal{F}$ is defined by a finite set of constraints $\boldsymbol{n}\,x \leq b$. Let $\mathcal{K}_\mathcal{F}$ be the set of pairs $(\boldsymbol{n}, b)$ representing these constraints. Function $x \mapsto \phi(x, \boldsymbol{u})$ is a continuous on $\mathcal{F}$ because we can write:

$$\phi(x, \boldsymbol{u}) = \min \left\{ \frac{b - \boldsymbol{n}\,x}{\boldsymbol{n}\,\boldsymbol{u}} \quad \text{for all } (\boldsymbol{n}, b) \in \mathcal{K}_\mathcal{F} \text{ such that } \boldsymbol{n}\,\boldsymbol{u} > 0 \right\}$$

Function $x \mapsto \phi(x, \boldsymbol{u})$ is uniformly continuous because it is continuous on the compact $\mathcal{F}$.

Choose $\varepsilon > 0$ and let $x, y \in \mathcal{F}$. Let the maximum $f^*(x, \boldsymbol{u})$ be reached in $x + \lambda^* \boldsymbol{u}$ with $0 \leq \lambda^* \leq \phi(x, \boldsymbol{u})$. Since f is uniformly continous on compact $\mathcal{F}$, there is $\eta > 0$ such that $|f(x+\lambda^*\boldsymbol{u}) - f(y+\lambda'\boldsymbol{u})| < \varepsilon$ whenever $\|x - y + (\lambda^* - \lambda')\boldsymbol{u}\| < \eta(1 + \|\boldsymbol{u}\|)$. In particular, it is sufficient to have $\|x - y\| < \eta$ and $|\lambda^* - \lambda'| < \eta$. Since ϕ is uniformly continuous, there is $\tau > 0$ such that $|\phi(y, \boldsymbol{u}) - \phi(x, \boldsymbol{u})| < \eta$ whenever $\|x - y\| < \tau$. We can then select $0 \leq \lambda' \leq \phi(y, \boldsymbol{u})$ such that $|\lambda^* - \lambda'| < \eta$. Therefore, when $\|x - y\| < \min(\eta, \tau)$, $f^*(x, \boldsymbol{u}) = f(x + \lambda^* \boldsymbol{u}) \leq f(y + \lambda' \boldsymbol{u}) + \varepsilon \leq f^*(y, \boldsymbol{u}) + \varepsilon$.

By reversing the roles of x and y in the above argument, we can similary establish that $f^*(y, \boldsymbol{u}) \leq f^*(x, \boldsymbol{u}) + \varepsilon$ when $\|x - y\| \leq \min(\eta, \tau)$. Function $x \mapsto f^*(x, \boldsymbol{u})$ is therefore uniformly continuous on $\mathcal{F}$. □

B.4 Stochastic Witness Direction Search

Each iteration of the following algorithm randomly chooses a feasible witness direction and performs an optimization along this direction. The successive search directions u_t are randomly selected (step 2a) according to some distribution P_t defined on $\mathcal{U}$. Distribution P_t possibly depends on values observed before time t.

Stochastic Witness Direction Search (WDS)

1) Find an initial feasible point $x_0 \in \mathcal{F}$.

2) For each $t = 1, 2, \dots,$

[1] We believe that the converse of theorem 20 is also true.

2a) Draw a direction $\boldsymbol{u}_t \in \mathcal{U}$ from a distribution P_t

2b) If $\boldsymbol{u} \in \mathcal{D}_{x_{t-1}}$ and $\boldsymbol{u}_t' \boldsymbol{\nabla} f(x_{t-1}) > 0$,

$$x_t \leftarrow \operatorname{argmax} f(x) \text{ under } x \in \{x_{t-1} + \lambda \boldsymbol{u}_t \in \mathcal{F},\, \lambda \geq 0\}$$

otherwise

$$x_t \leftarrow x_{t-1}.$$

Clearly the Stochastic WDS algorithm does not work if the distributions P_t always give probability zero to important directions. On the other hand, convergence is easily established if all feasible directions can be drawn with non zero minimal probability at any time.

Theorem 22 *Let f be a concave function defined on a compact convex set $\mathcal{F}$, differentiable with continuous derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Stochastic WDS algorithm above. Further assume there is $\pi > 0$ such that $P_t(\boldsymbol{u}) > \pi$ for all $\boldsymbol{u} \in \mathcal{U} \cap \mathcal{D}_{x_{t-1}}$. All accumulation points of the sequence x_t are then solutions of problem* (B.1) *with probability* 1.

Proof We want to evaluate the probability of event Q comprising all sequences of selected directions $(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots)$ leading to a situation where x_t has an accumulation point x^* that is not a solution of problem (B.1).

For each sequence of directions $(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots)$, the sequence $f(x_t)$ is increasing and bounded. It converges to $f^* = \sup_t f(x_t)$. We have $f(x^*) = f^*$ because f is continuous. By Theorem 17, there is a direction $\boldsymbol{u} \in \mathcal{U}$ such that $f^*(x^*, \boldsymbol{u}) > f^*$ and $\phi(x^*, \boldsymbol{u}) > 0$. Let x_{k_t} be a subsequence converging to x^*. Thanks to the continuity of ϕ, f^* and $\boldsymbol{\nabla} f$, there is a t_0 such that $f^*(x_{k_t}, \boldsymbol{u}) > f^*$ and $\phi(x_{k_t}, \boldsymbol{u}) > 0$ for all $k_t > t_0$.

Choose $\varepsilon > 0$ and let $Q_T \subset Q$ contain only sequences of directions such that $t_0 = T$. For any $k_t > T$, we know that $\phi(x_{k_t}, \boldsymbol{u}) > 0$ which means $\boldsymbol{u} \in \mathcal{U} \cap \mathcal{D}_{x_{k_t}}$. We also know that $\boldsymbol{u}_{k_t} \neq \boldsymbol{u}$ because we would otherwise obtain a contradiction $f(x_{k_t+1}) = f^*(x_{k_t}, \boldsymbol{u}) > f^*$. The probability of selecting such a $\boldsymbol{u}_{k_t}$ is therefore smaller than $(1 - \pi)$. The probability that this happens simultaneously for N distinct $k_t \geq T$ is smaller than $(1 - \pi)^N$ for any N. We get $P(Q_T) \leq \varepsilon / T^2$ by choosing N large enough.

Then we have $P(Q) = \sum_T P(Q_T) \leq \varepsilon \left(\sum_T 1/T^2 \right) = K\varepsilon$. Hence $P(Q) = 0$ because we can choose ε as small as we want, We can therefore assert with probability 1 that all accumulation points of sequence x_t are solutions. □

This condition on the distributions P_t is unfortunately too restrictive. The Process and Reprocess iterations of the Online LaSVM algorithm (Section 4.2) only exploit directions from very specific subsets.

On the other hand, the Online LaSVM algorithm only ensures that any remaining feasible direction at time T will eventually be selected with probability 1. Yet it is challenging to mathematically express that there is no coupling between the subset of time points t corresponding to a subsequence converging to a particular accumulation point, and the subset of time points t corresponding to the iterations where specific feasible directions are selected.

This problem also occurs in the deterministic Generalized SMO algorithm (Section 2.1.2). An asymptotic convergence proof [Lin, 2001] only exist for the important case of the SVM QP problem using a specific direction selection strategy. Following [Keerthi and Gilbert, 2002], we bypass this technical difficulty by defining a notion of approximate optimum and proving convergence in finite time. It is then easy to discuss the properties of the limit point.

B.5 Approximate Witness Direction Search

Definition 3 *Given a finite witness family $\mathcal{U}$ and the tolerances $\kappa > 0$ and $\tau > 0$, we say that x is a $\kappa\tau$-approximate solution of problem* (B.1) *when the following condition is verified:*

$$\forall \boldsymbol{u} \in \mathcal{U}, \quad \phi(x, \boldsymbol{u}) \leq \kappa \quad \text{or} \quad \boldsymbol{u}' \boldsymbol{\nabla} f(x) \leq \tau$$

A vector $\boldsymbol{u} \in \mathcal{R}_n$ such that $\phi(x, \boldsymbol{u}) > \kappa$ and $\boldsymbol{u}' \boldsymbol{\nabla} f(x) > \tau$ is called a $\kappa\tau$-violating direction in point x.

This definition is inspired by assertion (*iii*) in Theorem 17. The definition demands a *finite* witness family because this leads to Proposition 23 establishing that $\kappa\tau$-approximate solutions indicate the location of actual solutions when κ and τ tend to zero.

Proposition 23 *Let $\mathcal{U}$ be a finite witness family for bounded convex set $\mathcal{F}$. Consider a sequence $x_t \in \mathcal{F}$ of $\kappa_t\tau_t$-approximate solutions of problem* (B.1) *with $\tau_t \to 0$ and $\kappa_t \to 0$. The accumulation points of this sequence are solutions of problem* (B.1).

Proof Consider an accumulation point x^* and a subsequence x_{k_t} converging to x^*. Define function

$$(x, \tau, \kappa) \quad \mapsto \quad \psi(x, \tau, \kappa, \boldsymbol{u}) = \left(\boldsymbol{u}' \boldsymbol{\nabla} f(x) - \tau\right) \max\{0, \phi(x, \boldsymbol{u}) - \kappa\}$$

such that $\boldsymbol{u}$ is a $\kappa\tau$-violating direction if and only if $\psi(x, \kappa, \tau, \boldsymbol{u}) > 0$. Function ψ is continuous thanks to Theorem 20, Proposition 21 and to the continuity of $\boldsymbol{\nabla} f$. Therefore, we have $\psi(x_{k_t}, \kappa_{k_t}, \tau_{k_t}, \boldsymbol{u}) \leq 0$ for all $\boldsymbol{u} \in \mathcal{U}$. Taking the limit when $k_t \to \infty$ gives $\psi(x^*, 0, 0, \boldsymbol{u}) \leq 0$ for all $\boldsymbol{u} \in \mathcal{U}$. Theorem 17 then states that x^* is a solution. □

The following algorithm introduces the two tolerance parameters $\tau > 0$ and $\kappa > 0$ into the Stochastic Witness Direction Search algorithm.

Approximate Stochastic Witness Direction Search

1) Find an initial feasible point $x_0 \in \mathcal{F}$.

2) For each $t = 1, 2, \ldots,$

   2a) Draw a direction $\boldsymbol{u}_t \in \mathcal{U}$ from a probability distribution P_t

   2b) If $\boldsymbol{u}_t$ is a $\kappa\tau$-violating direction,

   $$x_t \leftarrow \operatorname{argmax} f(x) \text{ under } x \in \{x_{t-1} + \lambda \boldsymbol{u}_t \in \mathcal{F},\ \lambda \geq 0\}$$

   otherwise

   $$x_t \leftarrow x_{t-1}.$$

The successive search directions u_t are drawn from some unspecified distributions P_t defined on $\mathcal{U}$. Proposition 26 establishes that this algorithm always converges to some $x^* \in \mathcal{F}$ after a finite number of steps, regardless of the selected directions $(\boldsymbol{u}_t)$. The proof relies on the two intermediate results that generalize a lemma proposed by [Keerthi and Gilbert, 2002] in the case of quadratic functions.

Proposition 24 *If $\boldsymbol{u}_t$ is a $\kappa\tau$-violating direction in x_{t-1},*

$$\phi(x_t, \boldsymbol{u}_t)\, \boldsymbol{u}_t' \boldsymbol{\nabla} f(x_t) = 0$$

Proof Let the maximum $f(x_t) = f^*(x_{t-1}, \boldsymbol{u}_t)$ be attained in $x_t = x_{t-1} + \lambda^* \boldsymbol{u}_t$ with $0 \leq \lambda^* \leq \phi(x_{t-1}, \boldsymbol{u}_t)$. We know that $\lambda^* \neq 0$ because $\boldsymbol{u}_t$ is $\kappa\tau$-violating and Proposition 14 implies $f^*(x_{t-1}, \boldsymbol{u}_t) > f(x_{t-1})$. If λ^* reaches its upper bound, $\phi(x_t, \boldsymbol{u}_t) = 0$. Otherwise x_t is an unconstrained maximum and $\boldsymbol{u}_t' \boldsymbol{\nabla} f(x_t) = 0$. □

Proposition 25 *There is a constant $K > 0$ such that*

$$\forall t \ , \quad f(x_t) - f(x_{t-1}) \quad \geq \quad K \, \|x_t - x_{t-1}\|$$

Proof The relation is obvious when $\boldsymbol{u}_t$ is not a $\kappa\tau$-violating direction in x_{t-1}. Otherwise let the maximum $f(x_t) = f^*(x_{t-1}, \boldsymbol{u}_t)$ be attained in $x_t = x_{t-1} + \lambda^* \boldsymbol{u}_t$.
Let $\lambda = \nu\lambda^*$ with $0 < \nu \leq 1$. Since x_t is a maximum,

$$f(x_t) - f(x_{t-1}) = f(x_{t-1} + \lambda^* \boldsymbol{u}_t) - f(x_{t-1}) \geq f(x_{t-1} + \lambda \boldsymbol{u}_t) - f(x_{t-1})$$

Let H be the maximum over $\mathcal{F}$ of the norm of the Hessian of f.
A Taylor expansion with the Cauchy remainder gives

$$\left| \ f(x_{t-1} + \lambda \boldsymbol{u}_t) - f(x_{t-1}) - \lambda \boldsymbol{u}_t' \boldsymbol{\nabla} f(x_{t-1}) \ \right| \leq \frac{1}{2} \lambda^2 \|\boldsymbol{u}_t\|^2 H$$

or, more specifically,

$$f(x_{t-1} + \lambda \boldsymbol{u}_t) - f(x_{t-1}) - \lambda \boldsymbol{u}_t' \boldsymbol{\nabla} f(x_{t-1}) \ \geq \ -\frac{1}{2} \lambda^2 \|\boldsymbol{u}_t\|^2 H$$

Combining these inequalities yields

$$f(x_t) - f(x_{t-1}) \geq f(x_{t-1} + \lambda \boldsymbol{u}_t) - f(x_{t-1}) \geq \lambda \boldsymbol{u}_t' \boldsymbol{\nabla} f(x_{t-1}) - \frac{1}{2} \lambda^2 \|\boldsymbol{u}_t\|^2 H$$

Recalling $\boldsymbol{u}_t' \boldsymbol{\nabla} f(x_{t-1}) > \tau$, and $\lambda \|\boldsymbol{u}_t\| = \nu \|x_t - x_{t-1}\|$, we obtain

$$f(x_t) - f(x_{t-1}) \geq \|x_t - x_{t-1}\| \left(\nu \frac{\tau}{U} - \nu^2 \frac{1}{2} DH \right)$$

where $U = \max_{\mathcal{U}} \|\boldsymbol{u}\|$ and D is the diameter of the compact convex $\mathcal{F}$.
Choosing $\nu = \min\left(1, \frac{\tau}{UDH}\right)$ then gives the desired result. □

Proposition 26 *Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$. The Approximate Stochastic WDS algorithm converges to some $x^* \in \mathcal{F}$ after a finite number of steps.*

Proof Sequence $f(x_t)$ converges because it is increasing and bounded. Therefore it satisfies Cauchy's convergence criterion:

$$\begin{aligned} &\forall\, \varepsilon > 0, \exists\, t_0, \forall\, t_2 > t_1 > t_0, \\ &\qquad f(x_{t_2}) - f(x_{t_1}) = \sum_{t_1 < t \leq t_2} f(x_t) - f(x_{t-1}) < \varepsilon \end{aligned}$$

Using Proposition 25, we can write:

$$\begin{aligned} &\forall\, \varepsilon > 0, \exists\, t_0, \forall\, t_2 > t_1 > t_0, \\ &\qquad \|x_{t_2} - x_{t_1}\| \leq \sum_{t_1 < t \leq t_2} \|x_t - x_{t-1}\| \leq \sum_{t_1 < t \leq t_2} \frac{f(x_t) - f(x_{t-1})}{K} < \frac{\varepsilon}{K} \end{aligned}$$

Therefore sequence x_t satisfies Cauchy's condition and converges to some $x^* \in \mathcal{F}$.

Assume this convergence does not occur in a finite time. Since $\mathcal{U}$ is finite, the algorithm exploits at least one direction $\boldsymbol{u} \in \mathcal{U}$ an infinite number of times. Therefore there is a strictly increasing sequence of positive indices k_t such that $\boldsymbol{u}_{k_t} = \boldsymbol{u}$ is $\kappa\tau$-violating in point $x_{k_t - 1}$. We have then $\phi(x_{k_t - 1}, \boldsymbol{u}) > \kappa$ and $\boldsymbol{u}' \boldsymbol{\nabla} f(x_{k_t - 1}) > \tau$. By continuity we have $\phi(x^*, \boldsymbol{u}) \geq \kappa$ and $\boldsymbol{u}' \boldsymbol{\nabla} f(x^*) \geq \tau$. On the other hand, Proposition 24 states that $\phi(x_{k_t}, \boldsymbol{u})\, \boldsymbol{u}' \boldsymbol{\nabla} f(x_{k_t}) = 0$. By continuity when $t \to 0$, we obtain the contradiction $\phi(x^*, \boldsymbol{u})\, \boldsymbol{u}' \boldsymbol{\nabla} f(x^*) = 0$. □

In general, Proposition 26 only holds for $\kappa > 0$ and $\tau > 0$. [Keerthi and Gilbert, 2002] assert a similar property for $\kappa = 0$ and $\tau > 0$ in the case of SVMs only. Despite a mild flaw in the final argument of the initial proof, this assertion is correct [Takahashi and Nishi, 2003].

Proposition 26 does not prove that the limit x^* is related to the solution of the optimization problem (B.1). Additional assumptions on the direction selection step are required. Theorem 27 addresses the deterministic case by considering trivial distributions P_t that always select a $\kappa\tau$-violating direction if such directions exist. Theorem 28 addresses the stochastic case under mild conditions on the distribution P_t.

Theorem 27 *Let the concave function f defined on the compact convex set $\mathcal{F}$ be twice differentiable with continuous second derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Approximate Stochastic WDS algorithm above. Assume that step (2a) always selects a $\kappa\tau$-violating direction in x_{t-1} if such directions exist. Then x_t converges to a $\kappa\tau$-approximate solution of problem* (B.1) *after a finite number of steps.*

Proof Proposition 26 establishes that there is t_0 such that $x_t = x^*$ for all $t \geq t_0$. Assume there is a $\kappa\tau$-violating direction in x^*. For any $t > t_0$, step (2a) always selects such a direction, and step (2b) makes x_t different from $x_{t-1} = x^*$. This contradicts the definition of t_0. Therefore there are no $\kappa\tau$-violating direction in x^* and x^* is a $\kappa\tau$-approximate solution. □

B.5.1 Example (SMO)

The SMO algorithm (Section 2.1.2) is[2] an Approximate Stochastic WDS that always selects a $\kappa\tau$-violating direction when one exists. Therefore Theorem 27 applies.

Theorem 28 *Let the concave function f defined on the compact convex set $\mathcal{F}$ be twice differentiable with continuous second derivatives. Assume $\mathcal{U}$ is a finite witness set for set $\mathcal{F}$, and let the sequence x_t be defined by the Approximate Stochastic WDS algorithm above. Let p_t be the conditional probability that $\boldsymbol{u}_t$ is $\kappa\tau$-violating in x_{t-1} given that $\mathcal{U}$ contains such directions. Assume that $\limsup p_t > 0$. Then x_t converges with probability one to a $\kappa\tau$-approximate solution of problem* (B.1) *after a finite number of steps.*

Proof Proposition 26 establishes that for each sequence of selected directions $\boldsymbol{u}_t$, there is a time t_0 and a point $x^* \in \mathcal{F}$ such that $x_t = x^*$ for all $t \geq t_0$. Both t_0 and x^* depend on the sequence of directions $(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots)$.

We want to evaluate the probability of event Q comprising all sequences of directions $(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots)$ leading to a situation where there are $\kappa\tau$-violating directions in point x^*. Choose $\varepsilon > 0$ and let $Q_T \subset Q$ contain only sequences of decisions $(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots)$ such that $t_0 = T$.

Since $\limsup p_t > 0$, there is a subsequence k_t such that $p_{k_t} \geq \pi > 0$. For any $k_t > T$, we know that $\mathcal{U}$ contains $\kappa\tau$-violating directions in $x_{k_t-1} = x^*$. Direction $\boldsymbol{u}_{k_t}$ is not one of them because this would make x_{k_t} different from $x_{k_t-1} = x^*$. This occurs with probability $1 - p_{k_t} \leq 1 - \pi < 1$. The probability that this happens simultaneously for N distinct $k_t > T$ is smaller than $(1-\pi)^N$ for any N. We get $P(Q_T) \leq \varepsilon / T^2$ by choosing N large enough.

Then we have $P(Q) = \sum_T P(Q_T) \leq \varepsilon \left(\sum_T 1/T^2\right) = K\varepsilon$. Hence $P(Q) = 0$ because we can choose ε as small as we want. We can therefore assert with probability 1 that $\mathcal{U}$ contains no $\kappa\tau$-violating directions in point x^*. □

[2]Strictly speaking we should introduce the tolerance $\kappa > 0$ into the SMO algorithm. We can also claim that [Keerthi and Gilbert, 2002, Takahashi and Nishi, 2003] have established proposition 26 with $\kappa = 0$ and $\tau > 0$ for the specific case of SVMs. Therefore theorems 27 and 28 remain valid.

B.5.2 Example (LaSVM)

The LaSVM algorithm (Section 4.2) is[3] an Approximate Stochastic WDS that alternates two strategies for selecting search directions: Process and Reprocess. Theorem 28 applies because $\limsup p_t > 0$.

Proof Consider a arbitrary iteration T corresponding to a Reprocess.
Let us define the following assertions:

- A – There are τ-violating pairs (i, j) with both $i \in \mathcal{S}$ and $j \in \mathcal{S}$.
- B – A is false, but there are τ-violating pairs (i, j) with either $i \in \mathcal{S}$ or $j \in \mathcal{S}$.
- C – A and B are false, but there are τ-violating pairs (i, j).
- Q_t – Direction $\boldsymbol{u}_t$ is τ-violating in x_{t-1}.

A reasonment similar to the convergence discussion in Section 4.2 gives the following lower bounds (where n is the total number of examples).

$$\begin{array}{llll} P(Q_T|A) = 1 & & & \\ P(Q_T|B) = 0 & P(Q_{T+1}|B) \geq n^{-1} & & \\ P(Q_T|C) = 0 & P(Q_{T+1}|C) = 0 & P(Q_{T+2}|C) = 0 & P(Q_{T+3}|C) \geq n^{-2} \end{array}$$

Therefore

$$\begin{array}{l} P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid A\ ) \geq n^{-2} \\ P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid B\ ) \geq n^{-2} \\ P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid C\ ) \geq n^{-2} \end{array}$$

Since $p_t = P(Q_t \mid A \cup B \cup C)$ and since the events A, B, and C are disjoint, we have

$$p_T + p_{T+1} + p_{T+2} + p_{T+4} \geq P(\ Q_T \cup Q_{T+1} \cup Q_{T+2} \cup Q_{T+2} \mid A \cup B \cup C\ ) \geq n^{-2}$$

Therefore $\limsup p_t \geq \frac{1}{4}\, n^{-2}$. □

B.5.3 Example (LaSVM + Gradient Selection)

The LaSVM algorithm with Gradient Example Selection remains an Approximate WDS algorithm. Whenever Random Example Selection has a non zero probability to pick a τ-violating pair, Gradient Example Selection picks the a τ-violating pair with maximal gradient with probability one. Reasoning as above yields $\limsup p_t \geq 1$. Therefore Theorem 28 applies and the algorithm converges to a solution of the SVM QP problem.

B.5.4 Example (LaSVM + Active Selection + Randomized Search)

The LaSVM algorithm with Active Example Selection remains an Approximate WDS algorithm. However it does not necessarily verify the conditions of Theorem 28. There might indeed be τ-violating pairs that do not involve the example closest to the decision boundary.

However, convergence occurs when one uses the Randomized Search method to select an example near the decision boundary. There is indeed a probability greater than $1/n^M$ to draw a sample containing M copies of the same example. Reasonning as above yields $\limsup p_t \geq \frac{1}{4}\, n^{-2M}$. Therefore, Theorem 28 applies and the algorithm eventually converges to a solution of the SVM QP problem.

In practice this convergence occurs very slowly because it involves very rare events. On the other hand, there are good reasons to prefer the intermediate kernel classifiers visited by this algorithm (see Section 4.3).

[3] See footnote 2 discussing the tolerance κ in the case of SVMs.

C

Learning to Disambiguate Language Using World Knowledge

Disclaimer

This appendix presents an original work which is not directly related to the general topic of this thesis. However it introduces some of the first methods and results which have been produced following the ideas developed in the conclusion (Section 7.2.2). Hence, we believe this can be of some interest for the reader. This project is a joint work with Jason Weston, Nicolas Usunier and Ronan Collobert.

Abstract

We present a general framework and learning algorithm for the task of *concept labeling*: each word in a given sentence has to be tagged with the unique physical entity (e.g. person, object or location) or abstract concept it refers to. We show how grounding language using our framework allows *world knowledge* to be used during learning and prediction. We show experimentally using a simulated environment of interactions between actors, objects and locations that we can *learn* to use *world knowledge* to resolve ambiguities in language, such as word senses or reference resolution, without the use of *hand-crafted rules or features*.

C.1 Introduction

Much of the focus of the natural language processing community lies in solving syntactic or semantic tasks with the aid of sophisticated machine learning algorithms and the encoding of linguistic prior knowledge. For example, a typical way of encoding prior knowledge is to hand-code syntax-based input features for a given task. One of the most important features of natural language is that its real-world use (as a tool for humans) is to communicate something about our physical reality or metaphysical considerations of that reality. This is strong prior knowledge that is simply ignored in most current systems.

For example, in current parsing systems there is no allowance for the ability to disambiguate a sentence given knowledge of the physical reality of the world. So, if one happened to know that Bill owned a telescope while John did not, then this should affect parsing decisions given the sentence "John saw Bill in the park with his telescope." Likewise, in terms of reference resolution one could disambiguate the sentence "He passed the exam." if one happens to know that Bill is taking an exam and John is not. Further, one can improve disambiguation of the word bank in

"John went to the bank" if you happen to know whether John is out for a walk in the countryside or in the city. In summary, many human disambiguation decisions are in fact based on whether the current sentence agrees well with one's current *world model*. Such a model is dynamic as the current *state of the world* (e.g. the existing entities and their relations) changes over time.

In this paper, we propose a general framework for learning to use *world knowledge* called the *concept labeling* task. The knowledge we consider is rudimentary and can be viewed as a database of physical entities existing in the world (e.g. people, locations or objects) as well as abstract concepts, and relations between them, e.g. the location of one entity can be expressed in terms of its relation with another entity. Our task thus consists of labeling each word of a sentence with its corresponding *concept* from the database.

The solution to this task does not provide a full semantic interpretation of a sentence, but we believe is a first step towards that goal. Indeed, in many cases, the meaning of a sentence can only be uncovered after knowing exactly which concepts, e.g. which unique objects in the world, are involved. If one wants to interpret "He passed the exam", one has to infer not only that "He" refers to a "John", and "exam" to a school test, but also exactly which "John" and which test it was. In that sense, concept labeling is more general than the traditional tasks like word-sense disambiguation, co-reference resolution, and named-entity recognition, and can be seen as a unification of them.

We then go on to propose a tractable algorithm for this task that can learn to use world knowledge and linguistic content of a sentence seamlessly *without the use of any hand-crafted rules or features*. This is a challenging goal and standard algorithms do not achieve it.

The experimental evaluation of our algorithm uses a novel simulation procedure to generate natural language and concept label pairs: the simulation generates an evolving world, together with sentences describing the successive evolutions. This provides large labeled data sets with ambiguous sentences without any human intervention. Experiments presented in Section C.6 demonstrate that our algorithm can learn to use world knowledge for word disambiguation and reference resolution when standard methods cannot. We then go on to show in Section C.7 that we can also learn in the case of (i) using only weakly annotated data and (ii) more realistic data annotated by humans from RoboCup commentaries [Chen and Mooney, 2008].

In summary, the main contributions of this paper are:

1. the definition of the *concept labeling task*, including how to define the world (the database of concepts) (Section C.3),

2. a tractable learning algorithm for this task (using either fully or weakly supervised data) that uses no prior knowledge of how the concepts are expressed in natural language (Section C.4 and Section C.7),

3. the definition of a simulation framework for generating data for this task (Section C.5).

Although clearly only a first step towards the goal of language understanding, which is AI complete, we feel our work is an original way of tackling an important and central problem. In a nut-shell, we show one can learn (rather than engineer) to resolve ambiguities using world knowledge, which is a prerequisite for further semantic analysis, e.g. for communication.

C.2 Previous Work

Our work concerns learning the connection between two symbolic systems: the one of natural language and the one, non-linguistic, of the concepts present in a database. Making such an association has been studied as the *symbol grounding problem* [Harnad, 1990] in the literature.

More specifically, the problem of connecting natural language to another symbolic system is called *grounded* (or *situated*) *language processing* [Roy and Reiter, 2005].

Some of the earliest works that used world knowledge to improve linguistic processing involved hand-coded parsing and no learning at all, perhaps the most famous being situated in blocks world [Winograd *et al.*, 1972]. More recent works on grounded language acquisition have focused on *learning* to match language with some other representation. Grounding text with a visual representation also in a blocks-type world was tackled [Feldman *et al.*, 1996] (see also [Winston, 1976]). Other works also use visual grounding [Thibadeau, 1986, Siskind, 1994, Yu and Ballard, 2004, Fleischman and Roy, 2007, Barnard and Johnson, 2005], or a representation of the intended meaning in some formal language [Zettlemoyer and Collins, 2005, Fleischman and Roy, 2005, Kate and Mooney, 2007, Wong and Mooney, 2007, Chen and Mooney, 2008].

Example applications of such grounding include using the multimodal input to improve clustering (with respect to unimodal input) (see e.g. [Siskind, 1994]), word-sense disambiguation [Barnard and Johnson, 2005, Fleischman and Roy, 2005], or to make the machine predict one representation given the other. For instance, [Chen and Mooney, 2008] learn to generate textual commentaries of RoboCup soccer simulations from a representation of the actions in first-order logic, and [Zettlemoyer and Collins, 2005] learns to recover logical representations from natural language queries to a database. Although these learning systems can deal with some ambiguities in natural language (or ambiguities in the target formal representation, see e.g. [Chen and Mooney, 2008]), the representations that they consider, to the best of our knowledge, do not take into account the changing environment.

Much work has also been done on knowledge representation itself, see [Russell *et al.*, 1995] for an introduction. In our work, we choose a simple database representation which we use as input to our learning algorithm. The focus of this paper is not on knowledge representation, we made the simplest possible choice to simplify the exposition of the rest of the paper.

Work using linguistic context, i.e. previously uttered sentences, also ranges from dialogue systems, e.g. [Allen, 1995], to co-reference resolution [Soon *et al.*, 2001]. We do not consider this type of contextual knowledge in this paper, however our framework is extensible to those settings.

C.3 The Concept Labeling Task

We consider the following setup. One must learn a mapping from a natural language sentence $x \in \mathcal{X}$ to its labeling in terms of *concepts* $y \in \mathcal{Y}$, where y is an ordered set of concepts, one concept for each word in the sentence[1], i.e. $y = (c_1, \ldots, c_{|x|})$ where $c_i \in \mathcal{C}$, the set of concepts.

To learn this task one is given training data triples $\{x_i, y_i, \mathcal{U}_i\}_{i=1,\ldots,m} \in \mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ where $\mathcal{U}_i$ is one's knowledge of, i.e. current model of, the world (which we term a "*universe*").

Universe We define the universe as a set of concepts and their relation to other concepts: $\mathcal{U} = (\mathcal{C}, \mathcal{R}_1, \ldots, \mathcal{R}_n)$ where n is the number of types of relation and $(\mathcal{R}_i)_j \subset \mathcal{C}^2$, $\forall i = 1, \ldots, n$, $j = 1 \ldots |\mathcal{R}_i|$.

The *universe* we consider is in fact nothing more than a relational database, where records correspond to concepts and each kind of interaction between concepts is a relation table. To make things concrete we now describe the template database we use in this paper.

[1] When a phrase, rather than a word, should be mapped to a single concept, only the head word is mapped to that concept, and the other words are labeled with the *empty* ("-") concept.

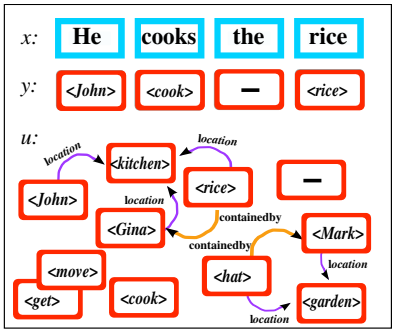


Figure C.1: **An example of a training triple** (x, y, u)**.** The universe u contains all the known concepts that exist, and their relations. The label y consists of the concepts that each word in the sentence x refers to, including the empty concept "-".

1. Each concept c of the database is identified using a unique string $name(c)$. Each physical object or action (verb) of the universe has its own referent. For example, two different cartons of milk will be referred to as $<milk1>$ and $<milk2>$[2].
2. We consider two relation tables[3] that can be expressed with the following formula:
   - $location(c) = c'$ with $c, c' \in \mathcal{C}$: the location of the concept c is the concept c'.
   - $containedby(c) = c'$ with $c, c' \in \mathcal{C}$: the concept c' physically holds the concept c.

An illustrating example of a training triple (x, y, u) is given in Figure C.1.

In our work, we only consider dynamic interactions i.e. in each relation table, relations can be inserted or deleted over time. Of course this setting is general, and one is free to define any database one wishes. For example one could (but in this paper we do not) encode static relations such as categories or hierarchies like the WordNet database [Miller, 1995]. The universe database u encapsulates the world knowledge available to the learner when making the predictions y about a sentence x, and a learning algorithm designed to solve the concept labeling task should be able to use the information within it.

Why is this task challenging? The main difficulty of this task arises with ambiguous words that can be mislabeled. Any tagging error would destroy subsequent semantic interpretation. A concept labeling algorithm must be able to use the available information to solve the ambiguities. In our work, we consider the following kinds of ambiguities (which of course, can be mixed within a sentence):

- **Location-based** ambiguities that can be resolved by the locations of the concepts. Examples: "The father picked *it* up" or "*He* got the coat in the hall". Information about the location of the father, co-located objects and so on can improve the accuracy of disambiguation.
- **Containedby-based** ambiguities that can be resolved through knowledge of *containedby* relations as in "the *milk* in the closet" or "the *one* in the closet" where there are several cartons of milk (e.g. one in the fridge and one in the closet).

[2]Here, we use understandable strings as identifiers for clarity reasons but they have no meaning for the system.
[3]Of course this list is easily expanded upon. Here, we give two simple properties of physical objects.

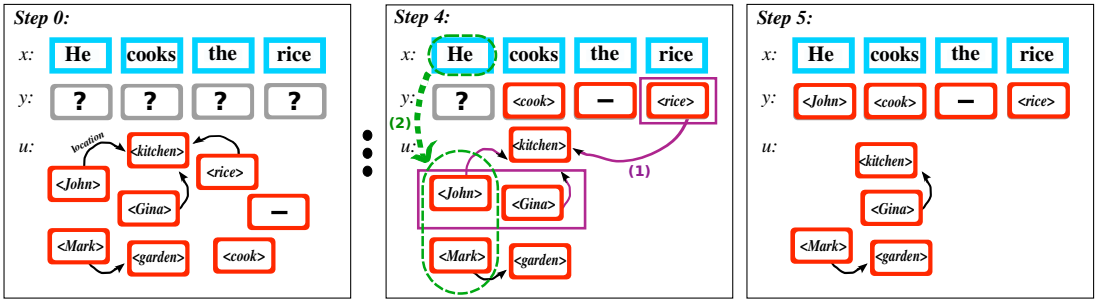


Figure C.2: **Inference Scheme.** Step 0 defines the task: find the concepts y given a sentence x and the current state of the *universe* u. For simplicity only relevant concepts and location relations are depicted. First, non-ambiguous words are labeled in steps 1-3 (not shown). In step 4, to tag the ambiguous pronoun "he", the system has to combine two pieces of information: (1) <*rice*> and the unknown concept might share the same location, <*kitchen*>, and, (2) "he" only refers to a subset of concepts in u (the males).

- **Category-based:** A concept is identified in a sentence by an ambiguous term (e.g. a pronoun, a polyseme) and the disambiguation can be resolved by using semantic categorization. Examples: "He cooks the rice in the kitchen" where both a male and a female are in the kitchen; "John drinks the *orange*" and "John ate the *orange*" where there are two objects <*orange fruit*> and <*orange juice*>, which can be disambiguated as one is drinkable and the other is eatable.

The first two kinds of ambiguities require the algorithm to be able to *learn* rules based on its available universe knowledge. The last kind can be solved using linguistic information such as word gender or category. However, the necessary rules or linguistic information are *not* given as input features and again the algorithm has to *learn* to infer them. This is one of the main goals of our work.

Figure C.2 describes how an algorithm could perform disambiguation. Even for a simple sentence the procedure is rather complex and somehow requires "reasoning". The next section describes the learning algorithm we propose for this task.

What is this useful for? A realistic setting where our approach can be applied immediately is within a computer game environment, e.g. multiplayer Internet games. Real-world settings are also possible but require, for example, vision technologies for building world knowledge beyond the scope of this work.

Our overall goal is to construct a semantic representation of a sentence. Concept labeling on its own is not sufficient to do this, however simply adding semantic role labeling (e.g. in the style of PropBank [Kingsbury and Palmer, 2002]) should then be sufficient. One would then know both the predicate concepts and the roles of other concepts with respect to those predicates. For example, "He cooks the rice" from Figure C.1 would be labeled with "He/ARG1 cooks/REL the/- rice/ARG2" as well as with the concept labels y. Predicting semantic roles should be straight-forward and has been addressed in numerous previous work [Collobert and Weston, 2008, Pradhan *et al.*, 2004]. For simplicity of exposition we therefore have not focused on this task.

Our system then has the potential to disambiguate examples such as the following: *"John went to the kitchen and Mark stayed in the living room. He cooked the rice and served dinner."*

The world knowledge that John is in the kitchen would come from the semantic representation predicted from the first sentence. This is used to resolve the pronoun "he" using further background knowledge that cooking is done in the kitchen. All of this inference is *learnt from examples*.

C.4 Learning Algorithm

Basic Argmax-type Inference A straight-forward approach one could adopt to learn a function that maps from sentence x to concept sequence y given u is to consider a model of the form:

$$y = f(x, u) = \text{argmax}_{y'} \;\; g(x, y', u), \tag{C.1}$$

where $g(\cdot)$ returns a scalar that should be a large value when the output concepts y' are consistent with both the sentence x and the current state of the universe u. To find such a function, one can choose a family of functions $g(\cdot)$ and pick the member which minimizes the error:

$$\sum_{i=1}^{m} L(y_i, f(x_i, \mathcal{U}_i)) \tag{C.2}$$

where the loss function L is 1 if its two arguments differ, and 0 otherwise. However, one practical issue of this choice of algorithm is that the exhaustive search over all possible concepts in equation (C.1) could be rather slow.

LaSO-type Inference In this paper we thus employ (a variation on) the LaSO (Learning As Search Optimization) algorithm [Daumé III and Marcu, 2005]. LaSO's central idea is to define a *search strategy*, and for each choice in the search path to use the function $g(\cdot)$ to make that choice. One then learns the function $g(\cdot)$ that optimizes the loss of interest, e.g. equation (C.2). Equation (C.1) is in fact a simple case of LaSO, with a simple (but slow) search strategy.

For our task we propose the following more efficient "order-free" search strategy: we greedily label the word we are most confident in (possibly the least ambiguous, which can be in any position in the sentence) and then use the known features of that concept to help label the remaining ones. That is, we perform the following steps:

1. For a given (x, u), start with predictions $\hat{y}_j^0 = \perp$, $j = 1, \ldots, |x|$, where $\perp$ means *unlabeled*.
2. On step t of the algorithm label greedily the concept with the highest score:

   $$\hat{y}^t = \text{argmax}_{y' \in S_t} \;\; g(x, y', u), \tag{C.3}$$

   where S_t is defined using $\hat{y}^{t-1}$ as follows:

   $$S_t = \bigcup_{j:\ \hat{y}_j^{t-1} = \perp} \{ y' \,|\, y'_j \in u \text{ and } \forall i \neq j, y'_i = \hat{y}_i^{t-1} \}$$

   That is, on each iteration one can label any thus far unlabeled word in the sentence with a concept; the algorithm picks the one it is most confident in.
3. Repeat (2) to label all words, i.e. $t = 1, \ldots |x|$.

Here, there are only $|u| \times |\, x|!$ computations of $g(\cdot)$, whereas equation (C.1) requires $|u|^{|x|}$ (and $|u| \gg |\, x|$).

Family of Functions Many choices of $g(\cdot)$ are possible. The actual form of $g(\cdot)$ we chose in our experiments is:

$$g(x, y, u) = \sum_{i=1}^{|x|} g_i(x, y_{-i}, u)^\top h(y_i, u) \tag{C.4}$$

where $g_i(\cdot) \in \mathbb{R}^N$ is a "sliding window" representation of width w centered on the i^{th} position in the sentence, y_{-i} is the same as y except that the i^{th} position $(y_{-i})_i = \perp$, and $h(\cdot) \in \mathbb{R}^N$ is a mapping into the same space as $g(\cdot)$. We constrain $||h(\perp, u)|| = 0$ so that as yet unlabeled outputs do not play a role.

A less mathematical explanation of this model is as follows: $g_i(\cdot)$ takes a window of the input sentence *and* previously labeled concepts centered around the i^{th} word and embeds them into an N dimensional space. $h(y_i, u)$ embeds the i^{th} concept into the same space, where both mappings are *learnt*. The magnitude of their dot product in this space indicates how confident the model is that the i^{th} word, given its context, should be labeled with concept y_i.

This representation is useful from a computational point of view because $g_i(\cdot)$ and $h(\cdot)$ can be cached and reused in equation (C.3), making inference fast.

We chose $g_i(\cdot)$ and $h(\cdot)$ to be simple two-layer linear neural networks in a similar spirit to [Collobert and Weston, 2008]. The first layer of both are so-called "Lookup Tables". We represent each word $\mathcal{W}$ in the dictionary with a unique vector $D(\mathcal{W}) \in \mathbb{R}^d$ and every unique concept name *name*(c) also with a unique vector $C(name(c)) \in \mathbb{R}^d$, where we *learn these mappings*. No hand-crafted syntactic features are used.

To represent a concept *and* its relations we do something slightly more complicated. A particular concept c (e.g. an object in a particular location, or being held by a particular person) is expressed as the concatenation of the three unique concept name vectors:

$$\bar{C}(c) = (C(name(c)), C(name(location(c))), C(name(containedby(c)))). \tag{C.5}$$

In this way, the learning algorithm can take these *dynamic* relations into account, if they are relevant for the labeling task. Hence, the first layer of the network $g_i(\cdot)$ outputs[4]:

$$g_i^1(x, y_{-i}, u) = \left(D(x_{i-\frac{w-1}{2}}), \ldots, D(x_{i+\frac{w-1}{2}}), \bar{C}((y_{-i})_{i-\frac{w-1}{2}}), \ldots, \bar{C}((y_{-i})_{i+\frac{w-1}{2}})\right)$$

The second layer is a linear layer that maps from this $4wd$ dimensional vector to the N dimensional output, i.e. overall we have the function:

$$g_i(x, y_{-i}, u) = W_g \; g_i^1(x, y_{-i}, u) + b_g.$$

Likewise, $h(y_i, u)$ has a first layer which outputs $\bar{C}(y_i)$, followed by a linear layer mapping from this $3d$ dimensional vector to N, i.e.

$$h(y_i, u) = W_h \; \bar{C}(y_i) + b_h.$$

Overall, we chose a linear architecture that avoids engineered features, assumes little prior knowledge about the mapping task in hand, but is powerful enough to capture many kinds of relations between words and concepts.

[4]Padding must be used when indices are out of bounds.

Training We train our system online, making a prediction for each example. If a prediction is incorrect an update is made to the model. We define the predicted labeling $\hat{y}^t$ at inference step t (see equation (C.3)) as y-good, compared to the true labeling y, if either $\hat{y}_i^t = y_i$ or $\hat{y}_i^t = \perp$ for all i. Then, during inference, if the current state in the search path $\hat{y}^t$ is no longer y-good we make an "early update" [Collins and Roark, 2004].

The update is a stochastic gradient step so that each possible y-good state one can choose from $\hat{y}^{t-1}$ is ranked higher than the current incorrect state, i.e. we would like to satisfy the ranking constraints:

$$g(x, \hat{y}^{t-1}_{+y_i}, u) > g(x, \hat{y}^t, u), \quad \{\forall i : \hat{y}_i^{t-1} = \perp\} \tag{C.6}$$

where $\hat{y}^{t-1}_{+y_i}$ denotes a vector which is the same as $\hat{y}^{t-1}$ except its i^{th} element is set to y_i. Note that if all such constraints are satisfied then all training examples must be correctly classified.

Why does this work? Consider again the example "He cooks the rice" in Figure C.2. We cannot resolve the first word in the sentence "He" with the true concept labeling <*John*> until we know that "rice" corresponds to the concept <*rice*> which we know is located in the kitchen, as is John, thereby making him the most likely referent.

This is why we choose to label words with concepts in an order independent of position in the sentence ("order-free") in Equation (C.3), e.g. we did not simply label from left to right because this does not work. The algorithm has to learn which word to label first, and presumably (and, this is what we have observed experimentally) it labels the least ambiguous words first. Once <*rice*> has been identified, its features including its location will influence the function $g(x, y, u)$ and the word "He" is more easily disambiguated.

Simultaneously, our method must learn the N dimensional representations $g_i(\cdot)$ and $h(\cdot)$ such that "He' matches with <*John*> rather than <*Gina*>, i.e. equation (C.4) is a larger value. This should happen because during training <*John*> and "He" often co-occur. This then concludes the disambiguation.

Note that our system can learn the general principle that two things that are in the same place are more likely to be referred to in the same sentence. Our system does not have to re-learn that for all possible places and things.

In general, our feature representation as given thus far makes it possible to resolve all kinds of ambiguities, both from syntax, semantics, or a combination. Indeed, all the cases given in Section C.3 are resolvable with our method.

C.5 A Simulation Environment

To produce a learning problem for our learning algorithm we define a simulation based on the framework defined in Section C.3. The goal of the simulation is to create an environment modeling a real world situation from which we can generate labeled training data. It has two components: (i) the definition of all the concepts constituting the environment and (ii) an iterative procedure that simulates activities within it *and* generates natural language sentences grounded by these actions.

C.5.1 Universe Definition

Our simulation framework is designed to be generic and easily adaptable to many environments. A universe is defined using two types of definition: (i) basic definitions shared by a large class of simulation instances; and (ii) definitions dedicated to a particular simulation.

Basic definitions This first part, shared by each simulation, implements all the tools to create and manipulate concepts and universes. This includes:

- Defines all the concepts corresponding to verbs in the language. Currently we have 15 verbs: <*move*>, <*get*>, <*give*>, <*put*>, <*sleep*>, <*wake up*>, <*play*>, <*drink*>, <*eat*>, <*bring*>, <*drop*>, <*sit*>, <*stand up*>, <*cook*>, <*work*>.
- Defines the relation types. Currently, the simulation implements *location*, *containedby*, *inherit* and *state*.
- Defines a function *exec*(c) for each verb c that takes as input a set of concepts (arguments) and the current universe u, and outputs a (modified) universe. This operation can potentially alter any relation that exists in the *universe*. For example the concept "<*move*>" could have a function *exec*(<move>) that takes two arguments: a physical object c'_1 and a location c'_2, and then outputs a universe where $location(c'_1) = c'_2$.
- Defines the function $(v, a) = event(u)$ which returns a randomly generated verb and set of arguments which are a *coherent* action given the universe. For example, it can return an actor moving or picking up an object. However, an actor cannot sit on a seat if it is occupied, give an object it does not have, and other similar intuitive constraints.
- Defines the function (x, y)=*generate*(v, a) which returns a sentence and concept labeling pair given a verb and set of arguments. This sentence should describe the event in natural language.

Environment definitions These definitions set up the specific physical environment for the chosen "world", i.e. the concepts (actors, objects and locations) that inhabit it. It defines the initial relations. From this starting point the simulation can then be executed.

C.5.2 Simulation Algorithm

The definitions above can create a universe. In order to generate training examples, it has to evolve, things must happen in it. To simulate activity in the artificial environment we iterate the following procedure:

1. Generate a new event, $(v, a) = event(u)$.
2. Generate a training sample, i.e. $generate(v, a)$.
3. Update the universe, i.e. $u := exec(v)(a, u)$.

Running this simple procedure modifies the universe at each step. For example, actors can change location and pick up, exchange or drop objects.

Step 2 is used to generate the training triple (x, y, u). Here, we have specified a computational method to generate a natural language sentence x. We define for each concept in u a set of phrases that can be used to name it (ambiguously or not). x is created by choosing and concatenating these terms along with linking adverbs, using a simple pre-defined grammar. Choosing how often to select ambiguous words at this step allows one to fix the rate of ambiguous terms in x. In our experiments we chose to forbid the generation of ambiguous sentences when the ambiguity cannot be resolved with the current universe information (as in "He drops an apple in the kitchen" when there is no way to guess who is "He", e.g. if there are several males holding apples in the kitchen).

This simulation makes testing learning algorithms straight-forward as one can control everything in it, from the size of its vocabulary to the amount of ambiguity. It also allows us to

x:	the father gets some yoghurt from the sideboard
y:	- <*John*> <*get*> - <*yoghurt*> - - <*sideboard*>
x:	he sits on the chair
y:	<*Mark*> <*sit*> - - <*chair*>
x:	she goes from the bedroom to the kitchen
y:	<*Gina*> <*move*> - - <*bedroom*> - - <*kitchen*>
x:	the brother gives the toy to her
y:	- <*Mark*> <*give*> - <*toy*> - <*Francoise*>
x:	the cat plays with it
y:	- <*cat*> <*play*> - <*ball*>

Table C.1: **Examples generated by the simulation.** Our task is to label a sentence x given only world knowledge u (not shown).

cheaply generate thousands of training examples in an online way without requiring any human annotation to test how algorithms scale. The particular environment we used for our experiments is described in the next section.

C.6 Experiments

Simulated World To conduct experiments on an environment with a reasonably large size we built the following artificial universe designed to simulate a house interior. It contains 58 concepts: the 15 verbs listed in Section C.5.1 along with 10 actors (<*John*>, <*dog*>,...), 15 small objects (<*water*>, <*chocolate*>, <*doll*>,...), 6 rooms (<*kitchen*>,...) and 12 pieces of furniture (<*couch*>, ...).

In our experiments, we define the set of describing words for each concept to contain at least two terms: an ambiguous one (using a pronoun) and a unique one. 75 words are used for generating sentences $x \in \mathcal{X}$. For example, an iteration of the procedure described in Section C.5.2 could produce the results:

1. The event <*move*>(<*Gina*>, <*hall*>) is picked.
2. Generate the sample $(x, y, u) =$ ("she goes from the bedroom to the hall", "<*Gina*> <*move*> - - <*bedroom*> - - <*hall*>", u).
3. Modify u with *location*(<*Gina*>) = <*hall*>.

This somewhat limited setup can still lead to millions of possible unique examples. Some examples of generated sentences are given in Table C.1. For our experiments we record 50,000 triples (x, y, u) for training and 20,000 for testing. Around 55% of these sentences contain lexical ambiguities.

Algorithms We compare several models. Firstly, we evaluate our "order-free" neural network based algorithm presented in Section C.4 (NN_{OF} using $x + u$) and the same where we *remove* the grounding to the *universe* (NN_{OF} using x).

The model *with* world knowledge has access to the *location* and *containedby* features of all concepts in the universe. For the model *without* world knowledge we remove the $C(name(location(c)))$

Method	Features	Train Err	Test Err
SVM_{struct}	x	42.26%	42.61%
SVM_{struct}	$x + u$ (*loc*, *contain*)	18.68%	23.57%
NN	x	35.80%	36.97%
NN_{LR}	x	32.80%	35.80%
NN_{LR}	$x + u$ (*loc*, *contain*)	5.42%	5.75%
NN_{OF}	x	32.50%	35.87%
NN_{OF}	$x + u$ (*contain*)	15.15%	17.04%
NN_{OF}	$x + u$ (*loc*)	5.07%	5.22%
NN_{OF}	$x + u$ (*loc*, *contain*)	**0.0%**	**0.11%**

Table C.2: **Medium-scale world simulation results.** We compare our order-free neural network (NN_{OF}) using world knowledge u to other variants: without world knowledge (x only), the same network using left-right resolution NN_{LF}, and SVM_{struct} versions. NN_{OF} using u performs best.

and $C(name(containedby(c)))$ features from the concept representation in equation (C.5) and are left with a pure tagging task, no different in spirit to tasks like named entity recognition.

In all experiments we used word and concept dimension $d = 20$, $g(\cdot)$ and $h(\cdot)$ have dimension $N = 200$, a sliding window width of $w = 13$ (i.e., 6 words on either side of a central word), and we chose the learning rate that minimized the training error given in equation (C.2). Complete code for our algorithms and simulations will be made available in time for the conference.

We also compare to other models. In terms of NNs, we compare order-free labeling to greedy left-to-right labeling (NN_{LR}) or only using a standard sliding window with no structured output feedback at all (NN). Finally, we compare all these models to a structured output SVM (SVM_{struct}) [Tsochantaridis *et al.*, 2005]. The features from the world model are just used as additional input features as in C.1. In this case, Viterbi is used to decode the outputs and all features are encoded in a binary format, as for the NN models. Only a linear model was used due to the infeasibility of training non-linear kernels (all the NN models are linear as well).

Results The results are given in Table C.2. The error rates, given by equation (C.2), express the proportion of sequences with *at least one* incorrect tag. They show that our model (NN_{OF}) learns to use world knowledge to disambiguate on this task: we obtain a test error close to 0% with this knowledge, and around 35% error without. The comparison with other algorithms highlights the following points: (i) order-free labeling of concepts is important compared to more restricted labeling schemes such as left-right labeling (NN_{LR}); (ii) the architecture of our NN which embeds concepts helps generalization; this should be compared to SVM_{struct} which does not perform as well. Note a nonlinear SVM *or* a linear SVM with hand-crafted features are likely to perform better, but the former is too slow and the latter is what we are trying to avoid in this work as such methods are brittle.

Table C.3 shows some of the features $C(name(c)) \in \mathbb{R}^d$ learnt by the model, analysing which concepts are similar to others using Euclidean distance in the 20-dimensional embedding space. We find that males, females, toys, animals, locations and actions are grouped together without giving this explicit information to the model. The model learns that these concepts are used in a similar context, e.g. the females are sometimes referred to by the word "she".

We constructed our simulation such that all ambiguities could be resolved with world knowledge, which is why we can obtain almost 0%: this is a good sanity check of whether our method is working well. That is, we believe it is a prerequisite that we do well on this problem if we

Query Concept	Most Similar Concepts
<*Gina*>	<*Francoise*>, <*Maggie*>
<*Mark*>	<*Harry*>, <*John*>
<*cat*>	<*hamster*>, <*dog*>
<*football*>	<*toy*>, <*videogame*>
<*chocolate*>	<*salad*>, <*milk*>
<*desk*>	<*bed*>, <*table*>
<*livingroom*>	<*kitchen*>, <*garden*>
<*get*>	<*sit*>, <*give*>

Table C.3: **Features learnt by the model.** Our model *learns* a representation of concepts in a 20 dimensional space. Finding nearest neighbors (via Euclidean distance) in this space we find similar concepts are close to each other. The model learns that female actors are similar, even though we have not given this information to the model.

hope to do well on harder tasks. The simulation we built uses rules to generate actions and utterances, however our learning algorithm uses no such hand-built rules but instead successfully *learns* them. We believe this flexibility is the key to success in real communication tasks, where brittle engineering approaches have been tried and failed.

One may still be concerned that the environment is so simple that we *know a priori* that the model we are learning is sufficiently expressive to capture all the relevant information in the world. In the real world one would never be able to achieve essentially zero (training/test) error. We therefore considered settings where aspects of the world could not be captured directly in the model that is learned: NN_{OF} using $x+u$ (*contain*) employs a world model with only a subset of the relational information (it does not have access to the *loc* relations). Similarly, we tried NN_{OF} using $x + u$ (*loc*) as well. The results in Table C.2 show that our model still learns to perform well (i.e. better than no world knowledge at all) in the presence of hidden/unavailable world knowledge.

Finally, if the amount of training data is reduced we can still perform well. With 5000 training examples for NN_{OF} ($x + u$(*loc*, *contain*)) with the same parameters we obtain 3.1% test error. This could probably be improved by reducing the high capacity of this model (d, N, w).

C.7 Weakly Labeled Data

So far we have considered learning from fully supervised data annotated with sequence labels of concepts explicitly aligned to words. Constructing such labeled data requires human annotation (as was done for example for Penn TreeBank or PropBank [Kingsbury and Palmer, 2002]).

Ideally, one would be able to learn from weakly supervised data of just observing language given the evolving world-state context. In this section we consider the weakly labeled case with exactly the same setting of training triples $\{x_i, y_i^*, \mathcal{U}_i\}_{i=1,\ldots,m}$ as before except y_i^* is a "bag" (set) of labels of size $|x_i|$ where there is no ordering/alignment information to the sentence x_i and show concept labeling can still be performed. This is a more realistic setting and is similar to the setting described in Chapter 6, except we learn to use world knowledge.

To do this, we employ the same inference algorithm with the same family of functions (C.4). The only thing that changes is the training algorithm. We still employ LaSo0 based learning but the update criteria is modified from (C.6) to the following ranking constraints:

$$g(x, \hat{y}^{t-1}_{+(i,j)}, u) > g(x, \hat{y}^t, u), \quad \{\forall i, j : \hat{y}_i^{t-1} = \perp \, , y_j^* \neq \perp \}$$

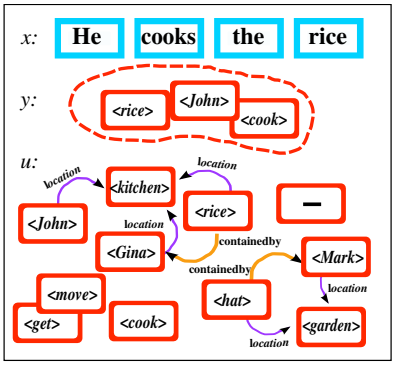


Figure C.3: **An example of a weakly labeled training triple** (x, y, u)**.** This setting is more realistic and does not require to create fully annotated training data.

where $\hat{y}^{t-1}_{+(i,j)}$ denotes a vector which is the same as $\hat{y}^{t-1}$ except its i^{th} element is set to y^*_j. After y^*_j is used in the inference algorithm it is set to $\perp$ so it cannot be used twice. Intuitively, if a label prediction for the word x_j in position j does not belong to the bag y^* then we require that any prediction that *does* belong to the bag y^* is ranked above this incorrect prediction. If all such constraints can be satisfied then we predict the correct bags. Even though the alignment (the concept labeling) is not given, this will implicitly learn it.

Simulation Result with Weak Labeling We employed this approach of weak labeling in an otherwise identical setup to the simulation experiments from Section C.6, i.e. we trained on triples $(x_i, y^*_i, \mathcal{U}_i)$ using both *loc* and *containedby* world knowledge. We obtained a concept labeling (alignment) training error of 0.64% and test error of 0.72% (using loss (C.2)). Note that the "bag" training error rate (percentage times we predict the correct bag) was 0%. This should be compared with the results in Table C.2 which were trained with fully supervised concept labeled data. We conclude that our method still performs very well in this more realistic weak setting.

RoboCup Commentaries We also tested our system on the RoboCup commentary data set available from `http://www.cs.utexas.edu/~ml/clamp/sportscasting/#data`. This data contains human commentaries on football simulations over four games labeled with semantic descriptions of actions (passes, offside, penalties, ... along with the players involved) extracted from the simulation, see [Chen and Mooney, 2008] for details. We treat this representation as a "bag" of concepts and train weak concept labeling. We trained on all unambiguous (sentence,bag-of-concept) pairs that occurred within 5 seconds of each other, training on only one match and testing on the other three, averaged over all four possible splits. We report the "matching" error [Chen and Mooney, 2008] which measures how often we predict the correct annotation for an ambiguous sentence. We do this by predicting the bag of labels and choosing to match to the bag from the ambiguous set that has the highest cosine similarity with our prediction. We achieve an F1 score of 0.669. Previously reported methods [Chen and Mooney, 2008] Krisper (0.645 F1) and Wasper-Gen (0.65 F1) achieve similar results, Wasper is worse (0.53 F1), while random matching yields 0.465 F1. In conclusion, results on this task indicate the usefulness of our method with weakly labeled human annotated data.

C.8 Conclusion

We have described a *general* framework for language grounding based on the task of *concept labeling*. The learning algorithm we propose is *scalable* and *flexible*: it learns with raw data, with no prior knowledge of how concepts in the world are expressed in natural language. We have tested our framework within a simulation, showing that it is possible to *learn* (rather than engineer) to resolve ambiguities using world knowledge. We also showed we can learn using only weakly supervised data and with real *human annotated* data (RoboCup commentaries). Although clearly only a first step towards the goal of language understanding we feel our work is an original way of tackling an important and central problem.

Many extensions are possible, e.g. further developing the simulation, predicting semantic roles for full semantic representation, and moving to an open domain. The most direct application of our work is probably for language understanding within a computer game, although potentially communication with any kind of static or mobile device (e.g. robots or cell phones) could apply.