Learning from Ambiguous Examples

13th May 2005
## Contents

1 Introduction ........................................ 4
   1.1 Learning From Ambiguous Examples .............. 4
   1.2 Image Retrieval Example .......................... 7
   1.3 Basics of Pattern Classification .................. 8
   1.4 Discussion ....................................... 11
   1.5 A Look Ahead ..................................... 13

2 Related Work ........................................ 15
   2.1 Definition ........................................ 15
   2.2 Learning Theory ................................... 16
   2.3 Learning Algorithms Based on Conditional Probability Density Functions .............. 16
   2.4 Learning Algorithms Based on Probability Density Functions ............................ 17
   2.5 Generalizations .................................... 18
   2.6 Related Work on Consistency ....................... 18
   2.7 Multi-label and Multi-class Classification ............. 19

3 Learning from Ambiguous Examples with Support Vector Machines ....................... 20
   3.1 Introduction ....................................... 20
      3.1.1 Maximum-Margin Formulation ................. 21
      3.1.2 Non-Separable Case ........................... 22
      3.1.3 Duality, Support Vectors & Kernelization .... 22
   3.2 Extending Support Vector Machines ................. 23
      3.2.1 Implicit Disambiguation ...................... 23
## CONTENTS

3.2.2 Multi-Instance Support Vector Machine Learning ........................................ 24  
3.2.3 Mixed-Integer Formulation ................................................................. 25  
3.2.4 Optimization Heuristics ................................................................. 27  
3.3 Experiments ................................................................. 27  
3.3.1 MUSK Data Set ................................................................. 28  
3.3.2 Automatic Image Annotation ................................................................. 28  
3.3.3 Text Categorization ................................................................. 29  
3.4 Discussion ................................................................. 30  

4 Learning from Ambiguous Examples with Linear Programming Boosting 31  
4.1 Introduction ................................................................. 31  
4.1.1 Linear Programming Formulation .................................................. 32  
4.1.2 Geometric Interpretation .................................................. 32  
4.1.3 Dual LP Formulation .................................................. 33  
4.1.4 Column Generation .................................................. 34  
4.2 Extending LPBoost ................................................................. 35  
4.2.1 Disjunctive Programming .................................................. 35  
4.2.2 Convexification .................................................. 36  
4.2.3 Parallel Reductions .................................................. 37  
4.2.4 Linearization .................................................. 39  
4.2.5 Duality .................................................. 40  
4.2.6 Row and Column Selection .................................................. 41  
4.3 Experiments ................................................................. 43  
4.3.1 Proof of Concept .................................................. 43  
4.4 Discussion ................................................................. 45  

5 Learning from Ambiguous Examples for Object Recognition 46  
5.1 Introduction ................................................................. 47  
5.2 Constellation Features ................................................................. 48  
5.2.1 LPQAP Approximation .................................................. 49  
5.3 Selecting Constellation Features ................................................................. 50  
5.3.1 LPBoost Revisited .................................................. 51
CONTENTS

5.3.2 Case 1: Single Part Constellation Features ........................................... 52
5.3.3 Disjunctive Programming Approximation .............................................. 53
5.3.4 Case 2: Multi-Part Constellation Features ............................................ 53
5.4 Experiments ................................................................................................. 55
5.5 Discussion ..................................................................................................... 56

6 Conclusions and Future Work ......................................................................... 57
  6.1 Learning from ambiguous examples with SVM ........................................... 57
  6.2 Learning from ambiguous examples with LPBoost ..................................... 57
  6.3 Learning from Ambiguous Examples for Object Recognition .................... 61

A Practical DPBoost Column Generation .......................................................... 67
  A.1 Convexification ............................................................................................ 67
  A.2 A New Heuristic for Column Generation .................................................... 68
    A.2.1 Dual LP ................................................................................................. 68
    A.2.2 Constraint Violation ............................................................................. 69

B List of Symbols Used .......................................................................................... 71
Chapter 1

Introduction

1.1 Learning From Ambiguous Examples

The burden of collecting labeled training data is the main impediment to widespread assimilation of supervised learning techniques. This is because supervised learning algorithms depend on training data in which each individual instance is assigned to a category. The instance-category pair is an example of the concept to be learned. In many applications, however, it is more practical to collect labels that are less specific. In particular, it is often much easier to collect labels that relate to sets of instances in an ambiguous fashion. The label is ambiguous because it is the consequence of an unspecified member or subset of members from the set. In this context, there is a mismatch between the single provided label and the desired labeling of individual instances within the set.

Indeed, this occurs whenever data items have polymorphic representations yet only a single label is provided. Examples of polymorphic representations are:

1. multiple instances of the same modality
2. multiple segments from a part-whole decomposition
3. multiple samples from a stochastic process

These representations are called a multi-instances. The pair, consisting of a multi-instance and an associated label, is called an ambiguous example. The example is said to be ambiguous because the association between the label and the individual member instances is not specified.
In this context, each individual member instance belongs to a category, yet these categories are not observed. Nevertheless, it is assumed that a rule deterministically computes multi-instance labels based on the labels of its members. This rule is known as the label semantics and may vary depending on the application. In the simplest case, the labels associated with a multi-instance are inherited directly from its member instances. These semantics are called 1-of-$N$ semantics. In order to derive useful information from an ambiguous example requires inference on the individual member instance labels. Under 1-of-$N$ semantics, there are $N$ distinct interpretations of the ambiguous example. Each interpretation is constructed by pairing the given label with an individual member instance; the pair is a potential unambiguous example.

The motivation for this work is based on the observation that ambiguous labels are easy to observe or collect manually, while the collection of specific labels for each individual member instances is difficult. Typically there is a trade-off between the quantity and specificity of training data that can be obtained with fixed resources (patience, time and energy). In this context, one can collect either: 1) a smaller set of unambiguous examples, or 2) a larger set of ambiguous examples. This trade-off is not universal. In some contexts, it may not be possible to label the individual members. In others, it may not make sense to use ambiguous examples at all.

Allowing for ambiguity within a supervised learning context has several advantages. For example, the collection of training data can be greatly simplified. It is often the case that ambiguous annotations are readily available in a given domain. One can utilize existing databases categorizations which, by virtue of their manual construction, are often ambiguous. The specific association between a category and a data item may be trivial and direct for a human to infer, even though the annotation is not specific. Indeed, a multi-instance representation arises naturally in the context of relational databases [41]. Secondly, it is often more intuitive and efficient for a human to provide annotations that are ambiguous, rather than ones that are very specific. Finally, this framework allows us to hypothesize and explore different decompositions or sampling techniques, in addition to different label semantics, to best explain manually-derived annotations.

The question is: Can an algorithm learn the underlying categories from ambiguous examples? This thesis answers this question in the affirmative. The burden of collecting a small number of unambiguous examples may be supplanted by specialized algorithms that are able to learn from larger and readily available collections of ambiguous examples. To do this, an algorithms must infer the individual member instance labels. This inference process is called disambiguation and depends on the assumed label semantics.

We derive new and specialized algorithms for learning from ambiguous examples from existing supervised algorithms. The construction of these algorithms employ the following fundamental assumption and
Figure 1.1: In several domains, the association between a categorical label and the input signal is ambiguous. This ambiguity can be modeled as a 1-out-of-$M$ selection; each choice corresponds to a unique interpretation of the signal. In computer vision applications, the presence of an object in a scene is reflected by the appearance of salient features or regions, emerging by interpretation from unstructured clutter. In information retrieval, the topic of a document is inferred from the topics of individual chapters and paragraphs. In bioinformatics, the classification of molecular function depends critically on the partially observed structure of the molecule. In these domains, a systematic treatment of the ambiguity of an input is important for successful learning.

guiding principle:

**Consistency of Disambiguation:** The true disambiguation of ambiguous examples is maximally consistent.

In other words, we assume that the quality of the classifier as evaluated by a supervised algorithm is higher under the true disambiguation than under any other disambiguation of the training set.
1.2 Image Retrieval Example

Students at a design school in Rhode Island are provided with means to stimulate their creativity. One resource that the school provides to support their endeavors consists of an extensive archive of photographs and other artwork collected from current magazines. Images in the collection are indexed according to their content, or subject matter. Categories are arranged hierarchically from general to specific. For example “tiger” and “zebra” categories are situated below “animals”. Sometimes distinct categories overlap, or similar categories appear across the hierarchy; although, for the purposes of this example we ignore these phenomena. Maintaining this collection requires several full-time librarians. These employees have the exclusive privilege of adding new images and rely on their own judgements of dominant subject matter to do so. As a result of the careful indexing of this collection, a student who wishes to develop an artistic theme based on cats or rabbits can easily retrieve relevant source material and thus inspiration.

A computer-aided system for image categorization will be of great value in this context. The design of such a system may be based on the principles of supervised learning. The two tasks to be solved are learning and classification. The objective of the learning task is a general purpose classifier for a given category. Being supervised, the system relies on a collection of examples to define the category. In one implementation of the system, learning is repeated for all categories in the hierarchy. Once a category has been learned, new images may be classified as belonging to the category or not. In this manner, the system maintains an up-to-date collection.

It is important to design a system that is able to learn from and classify images according to image-level annotations, otherwise known as keywords. Image-level annotations are easy to collect for training purposes. These annotations may be inferred from a caption or other text associated with the image. Moreover, the keyword indexing scheme is the prevailing method for indexing image databases. In contrast, one might rely on specific geometric annotations within the image, such as keypoints and bounding boxes (author?) [26, 38, 51]. These approaches are more restrictive, and involve tedious and time-consuming manual labeling. They impose constraints on the geometric annotations a priori. Furthermore, to collect the geometric annotations, specialized software must be developed and librarians must be trained to use it.

Classifying images based on their subject matter is a difficult problem. In computer vision, the problem is called object or scene recognition. In data mining, it is called image retrieval. The recognition of images containing tigers, for example, is nontrivial due to the variations of tiger appearance and pose, as well as the variations in external viewing conditions that affect these images. A further confounding problem is that
CHAPTER 1. INTRODUCTION

Figure 1.2: The role of segmentation in recognition. The average color of pixels in each segment determines the displayed color.

several objects beside a tiger may appear in the image. The image may also contain grass, trees, water, or in some cases maybe, a zebra. Arguably, the only way to “recognize” the tiger is to focus on the image segment occupied by the tiger. In other words, recognition depends on segmentation. This is accomplished by an image segmentation algorithm.

A simple color histogram may be used as a representation of segments. Figure 1.2 demonstrates the role of segmentation in recognition, by comparing color histograms of different image segments. Observe that color is a suitable predictor for the tiger when the segmentation is correct. Given an underlying classifier for the segments based on their color histograms, we can derive a classifier for the image. After automatic segmentation, the image is classified as containing a tiger if and only if there exists a tiger-classified segment (1-of-\(N\) tiger semantics).

For this application, we consider a polymorphic representation of an image based on a automatically generated part-whole decomposition. Color histograms are used to describe the individual segments. The image inherits the distinct labels of its member instances according to 1-of-\(N\) label semantics. With the ability to learn from ambiguous examples, a classifier for individual segments may be learned from a training set consisting of images and keywords.

1.3 Basics of Pattern Classification

Pattern classification addresses many of the questions that arise when studying the trends in a large collections of information, in particular, questions of association. Associations are mappings between inputs and outputs, and a rule for computing the mapping is called a classifier. Much attention in pattern classification research concerns the construction of classifiers in an automatic fashion in the context of supervised learning. To see how supervised methods can be generalized to ambiguous examples, we introduce some basic concepts and notation.
CHAPTER 1. INTRODUCTION

Representation

We assume that inputs may be encoded digitally in some fashion. Let \( I \in \mathbb{I} \) denote raw encoding of the input. In some cases, the encoding is captured using domain specific sensors such as a digital camera, a digital video camera, a mass spectrometer, etc. Otherwise, an existing encoding of the object may be used directly. A preprocessing stage converts the raw digital encoding into a feature vector. The space of feature vectors \( X \) is called the input space or feature space and we denote by \( x \in X \) the preprocessed feature vector for input \( I \). The non-trivial task of selecting meaningful features to represent objects is called feature engineering. Similarity between objects is measured in terms of these features. When the input features span a vector space with a Euclidean metric, the inner product from this space may be used. It is also possible to use non-metric representations, and to define a measure of similarity directly. The output of the association is an integer indicating the category to which the object belongs. The set of possible labels is called the output space and is denoted \( Y \). For binary classification, \( Y = \{-1, 1\} \). Let \( y \in Y \) denote the label. Using these notations, an example of a concept is denoted \((x, y)\).

To contrast with the standard case, inputs having polymorphic representations are considered. In some cases, the polymorphism exists because the raw input consists of multiple digital encodings from one sensor or multiple digital encodings from separate sensors of the same modality. Otherwise, the polymorphism is the result of preprocessing steps that decompose or draw samples from the raw input. In either case, a preprocessing stage converts the input \( I \) into a set of instances called a multi-instance and denoted \( X \equiv X(I) = \{x_1, \ldots, x_K\} \in 2^X \). The individual member instances \( x \in X \), \( \forall x \in X(I) \). The number of members \( K = K(I) \) may vary across inputs. Individual candidates \( x_1, \ldots, x_K \) have labels \( y_1, \ldots, y_K \) where \( y_k \in Y \), however these are unobserved. This set of labels is denoted \( \{y_1, \ldots, y_K\}_X \). In general, the observed output consists of a set of derived labels \( Y \equiv \{y_1, y_2, \ldots\} \in 2^Y \). Note that \( Y \) and \( \{y_1, \ldots, y_K\}_X \) are distinct but related label sets. The label semantics is a first-order predicate \( \mathcal{L} : 2^Y \times Y \to \{0, 1\} \) for computing the set of multi-instance labels \( Y \) from the individual member instance labels \( y_1, \ldots, y_K \). Binary values 0, 1 are used to represent truth values. When the predicate is satisfied \( \mathcal{L} (\{y_1, \ldots, y_K\}_X, y) = 1 \) the label \( y \) is a member of \( Y \). Using the 1-of-\( N \) semantics described previously, the predicate \( \mathcal{L} (\{y_1, \ldots, y_K\}_X, y) \) is equivalent to \( \exists \hat{y}, (\hat{y} \in \{y_1, \ldots, y_K\}_X) \land (\hat{y} = y) \). Labels in \( Y \) may result from a predicate involving more than one label. An ambiguous example is denoted \((X, Y)\).
CHAPTER 1. INTRODUCTION

Supervised Learning

*Supervised learning* refers to a class of methods for pattern classification that derive a classifier from a finite set of unambiguous input-output examples \( D = \{(x_i, y_i)\}, \ i = 1, \ldots, m \). This is also known as *learning by example* and *inductive inference*. Binary classification, where \( Y \) consists of two labels, is most prevalent. Since there are ways to reduce other classification problems to the binary case, it is arguably of primary importance. To reflect the stochastic nature of the world, examples are assumed to be independently and identically distributed according to a stationary but unknown probability distribution \((x, y) \sim p(X, Y)\).

It is assumed that a subset of features is sufficient to discriminate classes to some reasonable degree. The goal is to produce a classifier that will generalize; in other words produce the correct associations \( y \) for previously unseen inputs \( x \) in accordance with \( p(X, Y) \). Several popular learning algorithms are derived from the principle of risk minimization (author?) [50]. A risk minimization approach formulates the learning objective as an optimization problem and specifies an algorithm for solving it. Design issues include the mathematical specification of a discriminant and classifier, and the loss function that penalizes prediction errors. Various performance measures are used to quantify the efficacy of the learning.

Semi-Supervised Learning

*Semi-supervised learning* refers to a class of methods for pattern classification that derive a classifier from a finite set of input-output examples \( D = \{ (X_i, Y_i) \}, \ i = 1, \ldots, m \) that are incompletely specified. Learning from ambiguous examples is considered to be semi-supervised learning because the labels of individual member instances within each input are not specified. An ambiguous example is less informative than a typical example. To reflect the stochastic nature of the world, examples are assumed to be independently and identically distributed according to a stationary but unknown probability distribution \((X_i, Y_i) \sim p(2^X, 2^Y)\).

It is assumed that a subset of features is sufficient to predict labels of individual member instances and, by way of the label semantics, to predict labels of multi-instances with reasonable accuracy. As before, the goal is to produce a classifier that will generalize in accordance with \( p(2^X, 2^Y) \). The principle of consistent disambiguation is used to extend the risk minimization approaches used in supervised learning. Design issues include the mathematical specification of consistent disambiguation, and formulation of a tractable joint optimization problem.

**Learning from Ambiguous Examples**: The goal of learning from ambiguous examples is to induce a classifier for individual member instances from a finite training set of ambiguous
examples and a specification of the label semantics. The predictions of individual member instance labels will be biased to the task of correctly predicting the multi-instance labels which are determined via the label semantics.

The binary classification problem is a special case. In this case, the set of possible labels for member instances is $\mathbb{Y} = \{0, 1\}$. The set of possible labels for non-degenerate multi-instance is thus $\{0\}, \{1\}, \{0, 1\} \in 2^\mathbb{Y}$. Typically, the 1 label corresponds to a category of interest, while the 0 label corresponds to everything else (all other categories, background and outliers).

A specialization of particular importance is the multi-instance problem [18]. This formulation deals with ambiguous multi-instance inputs having binary labels using 1-of-$N$ semantics. Moreover, it is an asymmetric specialization of the binary classification case. This is because multiple-instance learning assumes binary input labels $\tilde{\mathbb{Y}}$ that correspond directly to the truth values $\tilde{\mathbb{Y}} \equiv \mathcal{L}([y_1, \ldots, y_K]_X, y)$, for $y$ equal to 1. This creates asymmetric ambiguity in the labels. Consider the ambiguous example $(X, \tilde{\mathbb{Y}})$. If $\tilde{Y} = 1$, then the true multi-instance label set $Y$ is either $\{1\}$, or $\{0, 1\}$. By first-order logic $\tilde{Y} = 1$ if and only if $\exists \hat{y} (\hat{y} \in \{y_1, \ldots, y_K\}_X) \land (\hat{y} = 1)$. On the other hand, if $\tilde{Y} = 0$ then the true multi-instance label set $Y$ is necessarily $\{0\}$. By first-order logic $\tilde{Y} = 0$ if and only if $\forall \hat{y} (\hat{y} \in \{y_1, \ldots, y_K\}_X) \land (\hat{y} = 0)$. The multi-instance label can therefore be expressed as a disjunction $\tilde{Y} = \bigvee_{\hat{y} \in \{y_1, \ldots, y_K\}_X} \hat{y}$ over the member instance labels.

**Multiple-Instance Learning:** The goal of multi-instance learning is to induce a classifier for individual member instances from a finite training set of ambiguous examples having binary (truth-value) labels. The 1-of-$N$ label semantics are used: the multi-instance label is a disjunction of member instance labels. In other words, a multi-instance label is true if and only if there exists a member instance label with a true label. The predictions of individual member instance labels will be biased to the task of correctly predicting the correct disjunction.

1.4 Discussion

A good representation is an essential ingredient for building a successful pattern classification system. In addition to distinguishing categories, the features of the chosen representation must be automatically computed from input sensor data. If the extracted features are too few, and unspecific, then objects from different classes will be indistinguishable. Conversely, if the features are too many, and too specific, then objects from
the same class will not be recognizable as such. Unless we are careful in our choice of representation for $x$, the feature space will be high-dimensional, the distribution $p(X,Y)$ will be complex and learning will require many examples. These observations are rigorously supported by learning theory (*add references*). In light of these observations, the importance of a systematic treatment of ambiguity become clear. Saying something is ambiguous does not mean that it is vague. Learning from ambiguous examples is a principled way to make specific and plausible inferences about ambiguous inputs in a pattern classification system.

Ambiguous signals occur frequently in pattern classification applications. This thesis considers applications in image retrieval, object recognition, bioinformatics, and document retrieval. A detailed example of ambiguous examples in an image retrieval application appears in this chapter. Further applications include prediction of: stock portfolio performance; epileptic periods (pre-attack, attack, post-attack); and protein-protein interactions, in addition to the identification of: people; landmarks; and protein superfamilies (*add references*). Finally, there is a growing interest of multiple-instance learning techniques in the areas of relational and multi-relational learning and inductive logic programming.

There are three common existing approaches for dealing with ambiguity. The first approach is to ignore it. By ignoring ambiguity, one must resort to a very simple and generic object representation. This is problematic, because a good representation is an essential ingredient for building a successful pattern classification system.

A second and alternative approach for dealing with ambiguity is based on the multi-instance representation. The idea is to define a similarity metric (kernel) for pairs of multi-instance inputs. Supervised learning algorithms are then applied directly using this similarity metric. One advantage of this approach is that research into kernel methods is very active and advanced. However, disambiguation is not handled explicitly, but rather implicitly within the similarity metric.

A third approach for dealing with ambiguity is to construct a probabilistic model in which the ambiguity is explicitly modeled. Building such models requires knowledge and experience.

Learning from ambiguous examples is a third method for dealing with ambiguity which retains the best of both worlds. Ambiguity is modeled explicitly using a polymorphic representation called a multi-instance. The design of multi-instances is intuitive and does not require probabilistic assumptions. Multi-instances contain candidate instances that are chosen to be highly specific and reflect dominant structures in a given input. Just as was the case with feature design, the design of multi-instance representations requires domain knowledge and is somewhat of an art.

Learning from ambiguous examples assumes label semantics. The semantics define a function mapping
from member instance labels to the multi-instance labels. The semantics also influence the inference procedure, whereby individual member instance labels are inferred from multi-instance labels. The ability to specify the label semantics in conjunction with the design of a multi-instance representation, allows one to explore different hypotheses for a predefined set of ambiguous labels.

The biggest challenge of this approach is the simultaneous disambiguation of ambiguous labels and inductive learning. The number of possible disambiguations for an entire training data set is exponential in the number of ambiguous examples. Disambiguation can be formulated of as a combinatorial search problem. If the cardinality of multi-instances is too large, then simultaneous disambiguation and inductive learning will be hopeless. To mitigate this problem, clustering and vector quantization may be used in order to reduce the cardinality of multi-instances. Furthermore, it was assumed that the training data set would be large. Therefore, large optimization problems are on the horizon.

As we have argued, a systematic treatment of ambiguity has the promise of extending the utility of traditional pattern classification techniques. This type of learning is included in the increasingly-prevalent area of semi-supervised learning, and is related to multiple-instance learning, transductive learning, and semi-supervised inductive learning. The significance of this learning framework is evident from the number and diversity of annotation-starved domains to which it is applied. Recently, a library of multi-instance learning algorithms was released [56]. It is open-source and written in Matlab.

1.5 A Look Ahead

In the following chapters, three algorithms for learning from ambiguous examples are proposed and compared. These are extensions of state-of-the-art supervised learning algorithms. In each case, the principle of inference by consistent disambiguation is employed.

The first algorithm, proposed in [3], is based on support vector machines. This algorithm seeks to disambiguate multi-instances while maximizing a measure of class separation by a hyperplane, known as the margin. This formulation results in a challenging (NP-hard) mixed-integer quadratic optimization problem, and local optima are obtained. Nevertheless, when compared on the standard multi-instance benchmark data sets (MUSK), the algorithm significantly outperforms all competitors except for one that was heavily engineered and tuned specifically for this benchmark. This is very encouraging. Because the approach is general, it works equally well in document and image retrieval domains.

The second algorithm, proposed in [1], is based on a linear programming formulation of supervised
learning. Our formulation is conceptually similar to that used above, however a boosting algorithm is used since the Representer Theorem (author?) [52] does not apply. The most important aspect of this formulation, however, is that it admits a convex lower bound. The bound is obtained using disjunctive programming. This helps to characterize the global optimality of a given disambiguation. Although the worst case space complexity is quartic in the size of the data set, recent experiments suggest that a sparse implementation of the algorithm will have a lower expected space complexity.

A third algorithm for learning from ambiguous examples is being developed. This approach improves upon that introduced in [2]. The primary application is component-based object recognition. The algorithm considers alternative label semantics which include logical and spatial interactions among members of a multi-instance. This demonstrates how the principles and techniques of learning from ambiguous examples may be generalized and successfully applied in domains with limited supervision. This also opens up new avenues for the development of semi-supervised learning algorithms.
Chapter 2

Related Work

2.1 Definition

The multi-instance learning problem was introduced by [18]. Their work was motivated by an application in drug-activity prediction. Automatic prediction of reactions is important for minimizing laboratory costs. It is known that molecular shape plays a significant role in determining reaction activity. Thus, the goal is to learn a classifier that can predict whether a new chemical compound will react with existing compounds in a desired manner based on molecular shape. Learning depends on a training set of existing compounds for which the reaction activity is known. This is a challenging problem because each molecule may exist in nature in one of several low-energy states known as conformations and these states have distinct shapes. Individual conformations are reactive or non-reactive. While the reaction activity for a molecule in the training set is known, the conformations that caused the reaction can not be observed.

Learning from multi-instance examples was first introduced by [18] to address this problem. They introduced the multi-instance representation to capture the polymorphic nature of the molecules. A feature vector representation is chosen to reflect the shape of a molecular conformation. The reaction activity of the molecule is assumed to depend on the reaction activities of the individual molecular conformations according to 1-of-\(N\) semantics. They present three learning algorithms which construct classifiers based on axis-parallel rectangles (APR) in feature space. The results obtained by these algorithms are among the most competitive today. Their multi-instance data set has become an official benchmark for this problem.
CHAPTER 2. RELATED WORK

2.2 Learning Theory

After the multi-instance problem was conceived a number of learning theoretic works were published on the topic. These included results that showed the problem was well-defined, and moreover could be solved. In [32], [5] and [11], theoretical arguments are given for the PAC-learnability of multi-instance concepts. Under the assumption that member instances are independent, a polynomial time algorithm is provided [32]. However, as shown in [5], without this assumption learning may be NP-hard.

2.3 Learning Algorithms Based on Conditional Probability Density Functions

Many supervised learning algorithms have been extended to deal with multi-instance inputs. The algorithms in this section are based on models of conditional probability density functions. In some cases, the modularity of these learning algorithms makes the extension to multi-instances relatively straight-forward. Only components that deal with the multi-instance inputs need to be changed.

In [53], two variants of the K-NN algorithm were developed. The most significant modification is the substitution of the Hausdorff metric for sets in place the instance-based metric normally used. The Hausdorff metric measures the distance between two sets of items, and is defined in terms of the individual instance-based distances. The application to multi-instances is immediate. In later work [24], a generalized measure of Hausdorff metric is used as well.

In [60], the authors extend decision trees. They do so by defining entropy and coverage functions for multi-instances instead of individual instances.

Neural network were first extended to the multi-instance setting by [43]. In their approach, the network outputs obtained on member instances are combined using a differentiable function that approximates the maximum of its inputs. Another application of neural network appears in [59]. The authors propose a modification of the error functional that takes into account the 1-of-\(N\) semantics of bags. No effort is made to achieve differentiability of the global error function.

More recently, [22, 23] developed a multi-instance kernel (similarity metric) and applied it to support vector machine learning. The similarity of multi-instances is defined in terms of the similarities of all possible pairs of members. One advantage of this approach is that research into kernel methods is very active and advanced.
Finally, in [20, 21], the authors propose a wrapper method used to extend an existing learning algorithm which 1) accepts instance weights, and 2) computes estimates of conditional label probabilities for individual instances. They express the conditional label probability of a multi-instance as the average of conditional label probabilities of member instances. Weights on the instances are used to balance the influence of individual member instances across all training examples. Their approach uses a symmetric treatment of positive and negative multi-instances. Using modifications when necessary, they apply this approach to extend support vector machines, decision trees, adaboost, logistic regression, nearest neighbor, and naive bayes learning algorithms. Of these, their method is best suited to logistic regression and Adaboost [21].

In [30], a Bayesian approach is adopted to learn a conditional label model. The conditional label model is expressed in terms of a sparse kernel expansion. Hidden variables are used for the member instance labels. A prior distribution over model parameters is specified and used to integrate out the uncertainty in the model parameters in the conditional model. The prior on member instance labels is truncated so that only valid instantiations have non-vanishing probability in accordance with the 1-of-\(N\) semantics of multi-instance labels. They employ a Metropolised blocked Gibbs sampler to approximate the otherwise intractable integral.

### 2.4 Learning Algorithms Based on Probability Density Functions

A separate line of research concerns the modeling of joint probability density functions.

Early work by [36, 35] learn a concept that consists of a single point in feature space. They define an “inverse generative” probability model through the specification of a likelihood over feature space that conditions on the training data. The diverse density is largest for points that are close to at least one member instance from each positive multi-instance and far from all member instances from all negative multi-instances. They use Bayes rule and the assumption that multi-instance inputs are independent. A noisy-or model to express the “at least one” notion probabilistically. A gradient based optimization procedure is used to find the solution.

Extending this work, [58, 57] derive EM-DD. They introduce hidden variables to indicate which member instance is responsible for the multi-instance label in accordance with the 1-of-\(N\) semantics. They use the EM algorithm to jointly maximize over the hidden variables and the target concept. The E-step selects member instances that are most likely given the model. The M-step optimizes the diverse density given the selection of member instances made in the E-step.

In the computer vision area of object recognition, the multi-instance problem appears in a probabilistic
setting where a hidden variable indexes the member instances of a multi-instance input. The authors of [19] define a generative probability model for an image, in terms of the appearance and relative location statistics of several interesting points. The image probability is expressed as a marginal over a hidden assignment variable \( h \) assigning keypoints from the image to foreground or background parts of the model. The likelihood ratio used for classification is expressed as a summation over \( h \) as well. Expectation maximization is used during learning, since the assignment \( h \) for each image is not observed.

### 2.5 Generalizations

In [42], the author argues that the multi-instance learning problem is the “natural frontier” between classical attribute-value learning and inductive logic programming (ILP). This is because all ILP problems can be converted into multi-instance problems, but typically can not be converted to attribute-value learning problems in a practical manner. For this reason, the multi-instance learning problem is viewed as an important extension of attribute-value learning.

In [46][48], the 1-of-\( N \) semantics of the multiple-instance learning problem are generalized to \( m \)-of-\( N \) semantics. The target concept is composed of a set of points. A multi-instance is labeled positively if and only if there is a subset of \( m \) target points that are each close to some point of the multi-instance. Furthermore, a set of repulsion points are used. None of the points in a positive bag are allowed to be located near the repulsion points. The Winnow algorithm used for learning a linear threshold encoding the concept. In [54], three further generalizations of the 1-of-\( N \) semantics are considered, including presence-based, threshold-based, and count-based multi-instance and multi-part concepts.

Several researchers have used multi-instance learning to solve ILP problems. The problems addressed can be significantly more general than that considered in the multi-instance setting, and include multi-part problems and multi-relational problems [14, 60, 13, 41, 39, 37]. In the domain of relational learning, a class of operations known as aggregations are defined to convert multi-instances into individual instances [41].

### 2.6 Related Work on Consistency

The notion of consistency has been considered in several semi-supervised learning contexts. In transductive learning, some of the examples have labels associated with them, while others do not and are therefore ambiguous. The task is to infer labels for the unlabeled examples. The problem is called semi-supervised due
to the incomplete specification of the training examples. The notion of a consistent disambiguation was used in this context by [28], and [9, 16]. In both cases, the theoretical justification is structural risk minimization. In [28], the author presents the underlying learning theory: the definition of hypothesis equivalence classes, the construction of a structure of increasing VC-dimension, the selection of an element from this structure via the maximization of a separating margin quantity over hyperplane and disambiguation parameters, and the relevance this has on the generalization error.

In [28], a mixed-integer formulation is presented that extends the quadratic formulation of the support vector machine. A heuristic rule for updating the unknown labels is interleaved with the update of hyperplane parameters using a support vector machine package. While it is shown that the algorithm converges after a finite number of iterations, it is only guaranteed to find local minima of the objective. Tests are performed on three text data sets.

A quadratic mixed-integer formulation based on the support vector machine is proposed in [9, 16]. This problem is solved to optimality for small data sets using standard mixed-integer solvers. To deal with large problem sizes, a localized version of the algorithm is developed. Performance is compared on several benchmark data sets. An additional non-convex formulation was considered and a heuristic algorithm was presented. Results were not conclusive in this case.

An alternative approach to the transductive learning problem appears in work by [8, 29]. In these approaches, the unlabeled examples in the training set are used to impose functional constraints on the classifier. However, there is no explicit disambiguation of the ambiguous examples. After learning the classifier may be used to classify the unlabeled instances in the training set.

2.7 Multi-label and Multi-class Classification

In general, learning from ambiguous examples can include multi-label and multi-class problems. Multi-class problems are classification problems where the outputs assume more than two different values. Multi-label problems are classification problems where the outputs can have several members. We focus on the binary classification case. The more general problems can be solved using binary classification methods.
Chapter 3

Learning from Ambiguous Examples
with Support Vector Machines

3.1 Introduction

The objective of support vector machine learning (SVM) is a linear discriminant function that robustly separates two categories in feature space $X \subseteq \mathbb{R}^d$. The theoretical foundation of SVM learning is based on empirical risk minimization. In this context, a bound on the generalization performance of the resulting classifier is derived. The bound depends on a measure of class separation by the linear discriminant function, known as the margin.

The linear discriminant function is defined in terms of a hyperplane that is parameterized by weight vector $w \in \mathbb{R}^d$ normal to the hyperplane and offset $b \in \mathbb{R}$, so that the points $x \in \mathbb{R}^d$ that lie on the hyperplane satisfy the linear equation $\langle w, x \rangle + b = 0$. The (signed) functional distance $\langle w, x \rangle + b$ is proportional to the perpendicular distance of a point from the hyperplane. This quantity is used to define a classifier $f(x) = \text{sgn}(\langle w, x \rangle + b)$. The true (signed) geometric distance of a point from the hyperplane is given by $\frac{\langle w, x \rangle + b}{\|w\|}$. Notice that the functional distance can be made arbitrarily large or small by scaling the weight vector.

The SVM takes as input a set of examples $D = \{(x_i, y_i), \ i = 1, \ldots, m\}$, where $x \in X$ and the output labels are binary $y \in Y = \{-1, 1\}$. The margin of an example $(x, y)$ is defined by $\gamma(x, y) = y (\langle w, x \rangle + b)$. The overall margin for the data set is $\gamma(D) \equiv \min_{1 \leq i \leq m} \gamma(x_i, y_i)$. The hyperplane defined by parameters
(w, b) separates the data set \( D \) if the margin quantities \( \gamma (x_i, y_i) \) are all positive; in other words \( \gamma (D) > 0 \). We say that \( D \) is linearly separable when there exists a hyperplane that separates \( D \).

If \( D \) is linearly separable, then quite often there are many separating hyperplanes. Which one should be used? According to the principle of empirical risk minimization [50], the best separating hyperplane is the one that achieves the largest margin.

### 3.1.1 Maximum-Margin Formulation

The maximum-margin separating hyperplane is found by solving the following optimization problem

\[
\text{Max-Margin} \quad \max_{w, b} \gamma (D) \quad (3.1)
\]

\[
\text{s.t.} \quad y_i (\langle w, x_i \rangle + b) \geq \gamma (D), \ \forall i, \quad (3.2)
\]

\[
\|w\| = 1. \quad (3.3)
\]

The weight vector is restricted to unit length so that the objective function is bounded. Notice that the variables of the optimization problem are used in the constraints, but do not appear in the objective. A more explicit expression of this optimization problem is obtained by introducing new variables defined as \( \hat{w} = w / \gamma (D) \) and \( \hat{b} = b / \gamma (D) \). Because the margin quantity \( \gamma (D) \) is not represented explicitly, the definition of \( \hat{w} \) and \( \hat{b} \) are implicit. Notice, in this case, that \( \|\hat{w}\| = \|w\| / \gamma (D) \) and therefore minimizing \( \|\hat{w}\| \) is equivalent to maximizing the margin subject to \( \|w\| = 1 \). The revised expression for the maximum-margin separating hyperplane is as follows

\[
\text{SVM-Primal (Hard Margin)} \quad \min_{\hat{w}, \hat{b}} \frac{1}{2} \|\hat{w}\|^2 \quad (3.4)
\]

\[
\text{s.t.} \quad y_i (\langle \hat{w}, x_i \rangle + \hat{b}) \geq 1, \ \forall i. \quad (3.5)
\]

This formulation called the SVM primal problem. It is evident that the problem is a quadratic programming problem, since the objective is quadratic and the constraints are linear. Because the original parameters \( (w, b) \) do not appear in this equivalent expression, we will drop the dots over parameters \( (\hat{w}, \hat{b}) \) in subsequent displays.
3.1.2 Non-Separable Case

In case no plane with positive margin $\gamma(D) > 0$ exists, the constraints of the SVM primal formulation cannot be met. To address this problem, a degree of flexibility is coded in the margin constraints by introducing *soft-margins*. An example $(x_i, y_i)$ may violate the margin constraint by an amount $\xi_i \geq 0$, called the *slack*, at the expense of adding a penalty proportional to $\xi_i$ to the objective. The modified primal SVM formulation is thus:

$$\text{SVM-Primal (Soft Margin)} \quad \min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} \xi_i$$  \hspace{1cm} (3.6)

s.t. $y_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \forall i,$ \hspace{1cm} (3.7)

$$\xi_i \geq 0, \forall i.$$ \hspace{1cm} (3.8)

3.1.3 Duality, Support Vectors & Kernelization

Appealing to Lagrangian techniques for constrained optimization the primal SVM problem with soft margins is re-formulated in dual form. The advantages of the dual formulation are: 1) a simplified set of constraints, 2) a parameterization that is readily interpretable, and 3) an optimization problem that admits kernelization [15][47].

The dual problem is expressed in terms of the dual variables $\alpha_i$, one for each training example.

$$\text{SVM-Dual} \quad \max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$ \hspace{1cm} (3.9)

s.t. $0 \leq \alpha_i \leq C, \forall i,$ \hspace{1cm} (3.10)

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$ \hspace{1cm} (3.11)

Kernelization is achieved by replacing the inner product $\langle x_i, x_j \rangle$ with a kernel function $K(x_i, x_j)$ satisfying Mercer’s condition [15][47]. The kernel emulates a mapping of all data points into a high dimensional space. Linear discriminants in the transformed space correspond to non-linear discriminants in the original space. Therefore, the same quadratic program for learning linear classifiers may be used to learn certain classes of non-linear classifiers. Further analysis of the primal-dual equivalence may be used to demonstrate that, typically, the solution of the dual SVM problem will be sparse; only a subset of the dual variables will be non-zero [15][47]. Points for which $\alpha_i > 0$ in the solution are called the *support vectors*, since they define or “support” the hyperplane.
3.2 Extending Support Vector Machines

The following presents an extension of support vector machine learning to the multi-instance learning problem. Recall that the goal of multi-instance learning is to induce a classifier for individual member instances from a finite training set of ambiguous examples having binary (truth-value) labels. The 1-of-N label semantics are used: the multi-instance label is a disjunction of member instance labels. In other words, a multi-instance label is true if and only if there exists a member instance label with a true label. The predictions of individual member instance labels will be biased to the task of correctly predicting the correct disjunction. In the multi-instance context, the training data set $D = \{(X_i, y_i), i = 1, \ldots, m\}$ consists of $m$ ambiguous examples consisting of multi-instances and binary labels representing truth-values. For ease of use with the SVM formulation, we adopt the convention that the labels $y_i \in \{-1, +1\}$ represent the truth values of false and true respectively. Recall that negative multi-instances are not ambiguous, since all member instances must have false labels $y = -1$.

The key to successful learning of a classifier from ambiguous examples is a process called disambiguation, the inference of individual member instance labels to explain the multi-instance label. Once the member instance labels are known, learning via SVM is easy. The difficulty is that disambiguation depends in a circular manner on the underlying classifier. It is here that we employ the fundamental assumption and guiding principle: the consistency of disambiguation. The principle states that the true disambiguation of ambiguous examples is maximally consistent. In other words, we assume that the quality of the classifier as evaluated by a supervised algorithm is higher under the true disambiguation than under any other disambiguation of the training set. In the case of support vector machine learning, the quantity of interest is the margin. In other words, we want to infer labels for member instances so that the margin is largest. A similar principle was used in large margin methods for transductive inference [28, 16]. They give a theoretical justification for this approach using the established principle of structural risk minimization [50].

3.2.1 Implicit Disambiguation

To extend the SVM formulation using the principle of consistent disambiguation, we define a multi-instance discriminant function which enforces it implicitly. The multi-instance discriminant function $F : 2^X \rightarrow \mathbb{R}$ is defined in terms of the linear discriminant function used in the SVM and has the following form
CHAPTER 3. LEARNING FROM AMBIGUOUS EXAMPLES WITH SUPPORT VECTOR MACHINES

\[ F(X) = \max_{x \in X} (\langle w, x \rangle + b). \] (3.12)

By virtue of taking the maximum operation over the member instances of \( X \) the multi-instance discriminant implicitly selects an individual member instance to determine its value. This member instance is called the “witness”. Predictions for multi-instance labels are computed by thresholding the discriminant \( \hat{y} = \text{sgn} \ F(X) \) as in the standard SVM context. In this manner the multi-instance discriminant performs disambiguation implicitly; the witness has the same label as the multi-instance label.

It is natural to define a functional multi-instance margin in terms of the multi-instance discriminant function

\[ \gamma = yF(X). \] (3.13)

This generalization reflects the fact that predictions for multi-instance labels take the form \( \hat{y} = \text{sgn} \ F(X) \). Notice that the multi-instance margin is defined by the margin of the witness. Therefore, it is sufficient for the witness alone to achieve a functional margin of \( \gamma \), in order for the multi-instance to achieve the same.

A multi-instance discriminant function separates the training data set \( D \) when \( \hat{y}_i = y_i \) for \( i = 1, \ldots, m \).

A hyperplane \( MI\text{-separates} \) a multi-instance data set if all member instances from negatively-labeled multi-instances lie on one side, while at least one member from each positively-labeled multi-instance lie on the other side. Since there may be several \( MI\)-separating hyperplanes, we adopt the principle used in section 3.1.1 and select the goal of learning is to find a separating hyperplane that also obtains the maximum multi-instance margin.

### 3.2.2 Multi-Instance Support Vector Machine Learning

Using the multi-instance discriminant function and margin we can apply maximum margin ideas to multi-instance learning. To do this, we simply replace the standard soft-margin constraint \( y (\langle w, x \rangle + b) \geq 1 - \xi \) in Equation 3.7, with the equivalent multi-instance version, which is stated here for clarity

\[ y \max_{x \in X} (\langle w, x \rangle + b) \geq 1 - \xi. \] (3.14)
The result of the substitution is the following quadratic programming problems with non-linear constraints

\[
\text{MI-SVM} \quad \min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} \xi_i \quad (3.15)
\]

s.t. \quad \begin{align*}
&y_i \max_{x \in X_i} (\langle w, x \rangle + b) \geq 1 - \xi_i, \forall i \quad (3.16) \\
&\xi_i \geq 0, \forall i. \quad (3.17)
\end{align*}

Notice that the linear discriminant function parameterized by \((w, b)\) is used twice: first, to order and select the witness, and subsequently, to determine the multi-instance margin based on this selection. Consistent disambiguation is self-reinforcing in this formulation. (*more explanation here*)

### 3.2.3 Mixed-Integer Formulation

As discussed in the introduction, the negatively-labeled multi-instances \((X_i, y_i)\) are not ambiguous. This fact is manifest in Equation (3.14). Setting \(y = -1\), we have obtain an equivalent set of margin constraints for the individual member instances

\[- \max_{x \in X_i} (\langle w, x \rangle + b) \geq (1 - \xi_i) \implies (\langle w, x \rangle + b) \leq - (1 - \xi_i), \forall x \in X_i \quad (3.18)\]

\[- (\langle w, x \rangle + b) \geq 1 - \xi_i, \forall x \in X_i \quad (3.19)\]

However, these constraints are tied together by a single slack variable \(\xi_i\) which complicates the implementation. For simplicity, we break apart negatively-labeled multi-instances, and treat the member instances as though they came from independent unambiguous training examples \(\{x_k\}, -1\) where \(x_k \in X_i, 1 \leq k \leq K_i\). The desired effect of this transformation is to decouple the constraints in Equation (3.19) through the introduction of additional slack variables. A second consequence of this transformation is the appearance of these additional slack variables in the objective in Equation (3.15). This can adversely affect the solution, especially when \(K_i\) are large. To alleviate this secondary effect, the penalty factor \(C\) for the newly created slack variables is scaled inversely proportional to \(K_i\).

For positively-labeled multi-instances, we introduce an integer disambiguation variable \(1 \leq z(i) \leq K_i\) to indicate the positive witness \(x_{z(i)} \in X_i\). If the disambiguation is consistent, then the multi-instance margin constraint from Equation (3.14) reduces to \(\langle w, x_{z(i)} \rangle + b \geq 1 - \xi_i\). Thus we arrive at the following equivalent
CHAPTER 3. LEARNING FROM AMBIGUOUS EXAMPLES WITH SUPPORT VECTOR MACHINES

Figure 3.1: Maximum margin MI-separating hyperplane. The MI-separating hyperplane is the solid line while the margins are indicated with dotted lines. Members of positively-labeled multi-instances are displayed as large circles with numbers $i$ that encode multi-instance membership $x \in X_i$, while members of negatively-labeled multi-instances are displayed as small gray circles. The individual member instances chosen as support vectors are bold. Notice that all members of negatively-labeled multi-instances obtain a large margin in the negative halfspace [down and to the left], and at least one member from each positively-labeled multi-instance ($i = 1, 2, 3$) obtains a large margin in the positive halfspace [up and to the right].

In this formulation, every positively-labeled multi-instance $X_i$ is thus effectively represented by a single member instance $x_{z(i)} \in X_i$. Notice that “non-witness” instances ($x_k \in X_i$ with $k \neq z(i)$) have no impact on the objective. For given disambiguation variables, the primal and dual objectives are identical to the standard SVM primal and Wolfe dual. The derivations also hold for general kernel functions $K$.

A simple example showing the maximum margin MI-separating hyperplane is illustrated, with explanation, in Figure 3.1.
3.2.4 Optimization Heuristics

As we have shown, learning from multi-instances can be cast as a mixed-integer program MI-SVM. In deriving optimization heuristics, we exploit the fact that for given integer variables, i.e. the disambiguation variables \( z(i) \) in MI-SVM, the problem reduces to a QP that can be solved exactly.

A general scheme for a simple optimization heuristic may be described as follows. Alternate the following two steps: (i) for given integer variables, solve the associated QP and find the optimal hyperplane, (ii) for a given hyperplane, update the integer variables in a way that (locally) minimizes the objective. The latter step may involve the update of a single integer variable \( z(i) \), or the simultaneous update of all integer variables. Since the integer variables are essentially decoupled given the hyperplane, this can be done very efficiently. Also notice that we can re-initialize the QP-solver at every iteration with the previously found solution, which will usually result in a significant speed-up. We initialize the scheme, by setting the witness \( x_{z(i)} \) equal to an imaginary member instance that is the centroid of the multi-instance \( X_i \). Algorithm 1 summarizes the algorithm in pseudo-code.

Algorithm 1: Pseudo-code for MI-SVM optimization heuristics (synchronous update).

1: \textbf{input:} training sample \( D \)
2: initialize \( x_{z(i)} = \sum_{x \in X_i} x / |K_i| \) for every positive input \( X_i \)
3: \textbf{repeat}
4: compute SVM solution \((w, b)\) for disambiguated training examples \( \{(x_{z(i)}, y_i), 1 \leq i \leq m\} \)
5: compute outputs \( f(x_k) = \langle w, x_k \rangle + b \) for all member instances \( x_k \in X_i, 1 \leq k \leq K_i \)
6: set \( z(i) = \arg \max_{1 \leq k \leq K_i} f(x_k) \)
7: \textbf{until} selector variables \( z(i) \) do not change
8: \textbf{output:} \((w, b)\)

There are many possibilities to refine the above heuristic strategy, for example, by starting from different initial conditions, by using branch and bound techniques to explore larger parts of the discrete part of the search space, by performing stochastic updates (simulated annealing) or by maintaining probabilities on the integer variables in the spirit of deterministic annealing. However, we have been able to achieve competitive results even with these simple optimization heuristics, which validate the maximum margin formulation of SVM. (*Further analysis may be included in the thesis.*)

3.3 Experiments

(*More explanation and editing needs to be done in this section.*)

We performed experiments on various data sets to evaluate the proposed techniques and compare them
Table 3.1: Accuracy results for various methods on the MUSK data sets.

<table>
<thead>
<tr>
<th>Method</th>
<th>MUSK1</th>
<th>MUSK2</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM-DD [58]</td>
<td>84.8</td>
<td>84.9</td>
</tr>
<tr>
<td>DD [36]</td>
<td>88.0</td>
<td>84.0</td>
</tr>
<tr>
<td>MI-NN [43]</td>
<td>88.9</td>
<td>82.5</td>
</tr>
<tr>
<td>IAPR [18]</td>
<td><strong>92.4</strong></td>
<td><strong>89.2</strong></td>
</tr>
<tr>
<td>mi-SVM</td>
<td>87.4</td>
<td>83.6</td>
</tr>
<tr>
<td>MI-SVM</td>
<td>77.9</td>
<td>84.3</td>
</tr>
</tbody>
</table>

Table 3.1: Accuracy results for various methods on the MUSK data sets.

to other methods for MIL. As a reference method we have implemented the EM Diverse Density (EM-DD) method [58], for which very competitive results have been reported on the MUSK benchmark.

3.3.1 MUSK Data Set

The MUSK data sets are the benchmark data sets used in virtually all previous approaches and have been described in detail in the landmark paper [18]. Both data sets, MUSK1 and MUSK2, consist of descriptions of molecules using multiple low-energy conformations. Each conformation is represented by a 166-dimensional feature vector derived from surface properties. MUSK1 contains on average approximately 6 conformation per molecule, while MUSK2 has on average more than 60 conformations in each bag. The averaged results of ten 10-fold cross-validation runs are summarized in Table 3.1. The SVM results are based on an RBF kernel $K(x, y) = \exp(-\gamma \|x - y\|^2)$ with coarsely optimized $\gamma$. For both MUSK1 and MUSK2 data sets, mi-SVM achieves competitive accuracy values. While MI-SVM outperforms mi-SVM on MUSK2, it is significantly worse on MUSK1. Although both methods fail to achieve the performance of the best method (iterated-APR).

3.3.2 Automatic Image Annotation

We have generated new MIL data sets for an image annotation task. The original data are color images from the Corel data set that have been preprocessed and segmented with the Blobworld system [12]. In this representation, an image consists of a set of segments (or blobs), each characterized by color, texture and shape descriptors. We have utilized three different categories ("elephant", "fox", "tiger") in our experiments.

---

1 However, the description of EM-DD in [58] seems to indicate that the authors used the test data to select the optimal solution obtained from multiple runs of the algorithm. In the pseudo-code formulation of EM-DD, $D_i$ is used to compute the error for the i-th data fold, where it should in fact be $D_i = D - D_i$ (using the notation of [58]). We have used the corrected version of the algorithm in our experiments and have obtained accuracy numbers using EM-DD that are more in line with previously published results.

2 Since the IAPR (iterated axis parallel rectangle) methods in [18] have been specifically designed and optimized for the MUSK classification task, the superiority of APR should not be interpreted as a failure; they compare favorably with other approaches to MIL.
In each case, the data sets have 100 positive and 100 negative example images. The latter have been randomly drawn from a pool of photos of other animals. Due to the limited accuracy of the image segmentation, the relative small number of region descriptors and the small training set size, this ends up being quite a hard classification problem. We are currently investigating alternative image representations in the context of applying MIL to content-based image retrieval and automated image indexing, for which we hope to achieve better (absolute) classification accuracies. However, these data sets seem legitimate for a comparative performance analysis. The results are summarized in Table 3.2. They show that both, mi-SVM and MI-SVM achieve a similar accuracy and outperform EM-DD by a few percent. While MI-SVM performed marginally better than mi-SVM, both heuristic methods were susceptible to other nearby local minima. Evidence of this effect was observed through experimentation with asynchronous updates, as described in Section 3.2.4, where we varied the number of integer variables updated at each iteration.

### 3.3.3 Text Categorization

Finally, we have generated MIL data sets for text categorization. Starting from the publicly available TREC9 data set, also known as OHSUMED, we have split documents into passages using overlapping windows of maximal 50 words each. The original data set consists of several years of selected MEDLINE articles. We have worked with the 1987 data set used as training data in the TREC9 filtering task which consists of approximately 54,000 documents. MEDLINE documents are annotated with MeSH terms (Medical Subject Headings), each defining a binary concept. The total number of MeSH terms in TREC9 was 4903. While we are currently performing a larger scale evaluation of MIL techniques on the full data set, we report preliminary results here on a smaller, randomly subsampled data set. We have been using the first seven categories of the pre-test portion with at least 100 positive examples. Compared to the other data sets the representation is extremely sparse and high-dimensional, which makes this data an interesting additional benchmark. Again, using linear and polynomial kernel functions, which are generally known to work well for text categorization, both methods show improved performance over EM-DD in almost all cases. No significant difference
CHAPTER 3. LEARNING FROM AMBIGUOUS EXAMPLES WITH SUPPORT VECTOR MACHINES

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Category</th>
<th>Dims inst/feat</th>
<th>EM-DD (rbf)</th>
<th>mi-SVM (linear)</th>
<th>mi-SVM (poly)</th>
<th>MI-SVM (linear)</th>
<th>MI-SVM (poly)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TST1</td>
<td></td>
<td>3224/6668</td>
<td>85.8</td>
<td>93.6</td>
<td>92.5</td>
<td>90.4</td>
<td>93.9</td>
</tr>
<tr>
<td>TST2</td>
<td></td>
<td>3344/6842</td>
<td>84.0</td>
<td>78.2</td>
<td>75.9</td>
<td>74.3</td>
<td>84.5</td>
</tr>
<tr>
<td>TST3</td>
<td></td>
<td>3246/6568</td>
<td>69.0</td>
<td>87.0</td>
<td>83.3</td>
<td>69.0</td>
<td>82.2</td>
</tr>
<tr>
<td>TST4</td>
<td></td>
<td>3391/6626</td>
<td>80.5</td>
<td>82.8</td>
<td>80.0</td>
<td>69.6</td>
<td>82.4</td>
</tr>
<tr>
<td>TST7</td>
<td></td>
<td>3367/7037</td>
<td>75.4</td>
<td>81.3</td>
<td>78.7</td>
<td>81.3</td>
<td>78.0</td>
</tr>
<tr>
<td>TST9</td>
<td></td>
<td>3300/6982</td>
<td>65.5</td>
<td>67.5</td>
<td>65.6</td>
<td>55.2</td>
<td>60.2</td>
</tr>
<tr>
<td>TST10</td>
<td></td>
<td>3453/7073</td>
<td>78.5</td>
<td>79.6</td>
<td>78.3</td>
<td>52.6</td>
<td>79.5</td>
</tr>
</tbody>
</table>

Table 3.3: Classification accuracy of different methods on the TREC9 document categorization sets.

between the two methods is clearly evident for the text classification task.

3.4 Discussion

This chapter presented a novel approach to multiple-instance learning based on a generalization of the maximum margin idea used in SVM classification. Although the formulation leads to a difficult mixed-integer optimization problems, even simple local optimization heuristics already yield quite competitive results compared to the baseline approach. We conjecture that better optimization techniques, that can for example avoid unfavorable local minima, may further improve the classification accuracy.

As far as the MIL research problem is concerned, we have considered a wider range of data sets and applications than is usually done and have been able to obtain very good results across a variety of data sets. These data sets are available to the public to encourage further empirical comparisons.
Chapter 4

Learning from Ambiguous Examples with Linear Programming Boosting

4.1 Introduction

The objective of linear programming boosting (LPBoost) is a linear discriminant function that robustly separates two categories in a suitably transformed feature space $\mathcal{X}' \subseteq \mathbb{R}^n$. The theoretical foundation of LPBoost learning is based on empirical risk minimization. In this context, a bound on the generalization performance of the resulting classifier is derived. The bound depends on a measure of class separation by the linear discriminant function, known as the margin.

The linear discriminant function is defined by an ensemble which is specified by a set of features and combination weights. The ensemble has the following form

$$f(x) = \sum_{k=1}^{n} \alpha_k h_k(x)$$

where $h_k : \mathcal{X} \rightarrow [-1, 1]$, $k = 1, \ldots, n$ are the features that are sometimes called weak hypotheses and $\alpha_k \geq 0$ are combination weights. The ensemble value is used to define a classifier $F(x) = \text{sgn} f(x)$.

LPBoost takes as input a set of examples $\mathcal{D} = \{(x_i, y_i), \ i = 1, \ldots, m\}$, where $x \in \mathcal{X}$ and the output labels are binary $y \in \mathcal{Y} = \{-1, 1\}$. The margin of an example $(x, y)$ is defined by $\gamma(x, y) = y \sum_{k=1}^{n} \alpha_k h_k(x)$. The overall margin for the data set is $\gamma(\mathcal{D}) \equiv \min_{1 \leq i \leq m} \gamma(x_i, y_i)$. The ensemble
defined by weights $\alpha_k$ and features $h_k$ separates the data set $\mathcal{D}$ if the margin quantities $\gamma(x_i, y_i)$ are all positive; in other words $\gamma(\mathcal{D}) > 0$. We say that $\mathcal{D}$ is linearly separable when there exists a hyperplane that separates $\mathcal{D}$.

If $\mathcal{D}$ is linearly separable, then quite often there are many separating ensembles. Which one should be used? According to the principle of empirical risk minimization [17], the best separating ensemble minimizes $\sum_{k=1}^{n} \alpha_k$. Moreover, if the training set is not separable, a soft-margin and associated slack variable $\xi_i$ is introduced to recover separability. In this case, the best separating ensemble minimizes $\sum_{k=1}^{n} \alpha_k + C \sum_{i=1}^{m} \xi_i$, where $C > 0$ controls the complexity of the resulting ensemble.

### 4.1.1 Linear Programming Formulation

Based on the above criterion, learning can be formulated directly as a linear program.

$$\min_{\alpha, \xi} \sum_{k=1}^{n} \alpha_k + C \sum_{i=1}^{m} \xi_i$$  \hspace{1cm} (4.2)

subject to

$$y_i \sum_{k=1}^{n} \alpha_k h_k(x_i) + \xi_i \geq 1 \quad \forall i$$  \hspace{1cm} (4.3)

$$\alpha_k, \xi_i \geq 0 \quad \forall k, i$$  \hspace{1cm} (4.4)

The objective is often called a 1-norm soft-margin objective. The constraints ensure that the ensemble - with the help of slack variables $\xi_i$ - obtains the desired minimum margin $\gamma = 1$ on each training example. The fact that $\alpha_k \geq 0$ is not a restriction assuming that the set of features $h_k$ is closed under negation. For a moment ignoring the fact that there are typically a very large number of weak hypotheses $h_k(x)$, one can solve the optimization using linear programming. The solution to this LP determines our ensemble classifier $f(x) = \sum_{k=1}^{n} \alpha_k h_k(x)$. Note that the formulation is valid even when all training examples have the same label, and may be used for dimension reduction and novelty detection applications [44].

### 4.1.2 Geometric Interpretation

To gain insight into the geometric structure of the feasible region of LPBoost, it is useful to introduce set notations that correspond to constraints in Equations (4.3) and (4.4). Let $Q = \{(\alpha, \xi) : \alpha, \xi \geq 0\}$ denote
Figure 4.1: LPBoost feasible region: Shaded region depict the feasible region defined by two training examples. Only 2 dimensions are shown. There are axes for each $\alpha_k$ and $\xi_i$.

the positive quadrant, and $\mathcal{H}_i \equiv \{ (\alpha, \xi) : y_i \sum_k \alpha_k h_k(x_i) + \xi_i \geq 1 \}$ denote the region of feasibility corresponding to the $i$-th example. Using these notations, the feasible region is expressed

$$ (\alpha, \xi) \in \mathcal{Q} \cap \bigcap_{i=1}^m \mathcal{H}_i. \quad (4.5) $$

A 2-dimensional projection of this region is depicted in Figure 4.1.

### 4.1.3 Dual LP Formulation

To characterize the solution of the LP, it is useful to consider the dual problem and conditions of complementarity as in [17]. The dual of Equation (4.2-4.4) can be written as

$$ \max_u \sum_{i=1}^m u_i \quad (4.6) $$

$$ \text{s.t.} \quad \sum_{i=1}^n u_i y_i h_k(x_i) \leq 1 \quad \forall k \quad (4.7) $$

$$ 0 \leq u_i \leq C \quad \forall i \quad (4.8) $$

The dual variables $u_i$ can be considered to be misclassification costs. Relating the dual and primal problems, the conditions of complementarity are as follows

$$ u_i (y_i f(x_i) + \xi_i - 1) = 0, \text{ and } \alpha_k \left( \sum_{i=1}^m u_i y_i h_k(x_i) - 1 \right) = 0. \quad (4.9) $$
Since the optimal values of the slack variables are implicitly determined by \( \alpha \) as \( \xi_i(\alpha) = [1 - y_i f(x_i)]_+ \), the first set of conditions states that \( u_i = 0 \) whenever \( y_i f(x_i) > 1 \). Examples with non-vanishing misclassification costs must have tight margin constraints. The second set of conditions ensures that \( \alpha_k = 0 \), if \( \sum_{i=1}^{m} u_i y_i h_k(x_i) < 1 \). Weak hypotheses with non-zero weights \( \alpha_k \) must satisfy \( \sum_{i=1}^{m} u_i y_i h_k(x_i) = 1 \).

Thus, a typical LPBoost ensemble solution may be sparse in two ways: 1) only a small number of weak hypothesis with \( \alpha_k > 0 \) will contribute to the ensemble, and 2) the solution will depend only on a subset of training examples with \( u_i > 0 \).

4.1.4 Column Generation

The simplex method for solving LP maintains a tableau representing the current basic feasible solution (bfs) plus additional values that are used to transition from one bfs to another. The tableau has as many rows as there are constraints in the LP, and as many columns as there are variables in the LP. When the number of variables is large, it is impractical, and moreover unnecessary, to store and update the entire tableau. The columns that correspond to the basic feasible solution are sufficient to compute the objective.

The practical solution to this problem is a technique called column generation (CG). CG solves a sequence of smaller problems using only a subset of the columns; variables of the excluded columns are fixed at zero. Hence, these are called restricted master problems. By duality, the solution of a restricted master problem will either correspond to the solution or an infeasible point of the original dual problem. In particular, constraints of the dual may be violated when their corressponding primal variables are constrained to zero. Once a restricted master problem is solved, the solution is analyzed to determine new columns to be considered. The criterion for adding a new column (variable) is the amount by which its corresponding dual constraint is violated. After selection, the restricted tableau is re-optimized with the selected columns.

LPBoost [17] employs CG to select weak hypotheses of the ensemble in rounds. The violation of dual constraints in Equation (4.7) is quantified by the following score function

\[
S(k) = \sum_{i=1}^{m} u_i y_i h_k(x_i) - 1 .
\]

LPBoost selects weak hypotheses indexed by \( k \) for which \( S(k) \) most positive.
4.2 Extending LPBoost

To extend LPBoost to multi-instance learning, we follow a similar development as was presented for extending support vector machines. The notion of a consistent disambiguation, and the definition of a multi-instance discriminant function and margin, are analogous, except that the inner product $\langle w, x \rangle$ is replaced with the ensemble output $\sum_{k=1}^{n} \alpha_k h_k(x)$.

**More background is needed here, although it will parallel what is described for SVM - a key difference is that we do not introduce integer variables, but instead relax a disjunctive set of constraints.**

4.2.1 Disjunctive Programming

In the support vector machine approach described above, we defined selector variables in order to recover a decoupled, two-stage mixed-integer optimization strategy. Stages alternated optimization over integer variables using local optimization heuristics, and continuous variables by solving a convex optimization problem.

In order to deal with multi-instance ambiguity in this setting, we employ the disjunctive programming framework [6, 31]. The authors propose the techniques of convexification and parallel reduction to generate a series of convex approximations to optimization problems over disjunctive sets. *More background is needed here*

In order to applying these principles, we compile the “unfolded” margin constraints for a positively-labeled multi-instance into a set of disjunctive constraints on $(\alpha, \xi)$. Denote by

$$H_i(x) \equiv \left\{ (\alpha, \xi) : y_i \sum_k \alpha_k h_k(x) + \xi_i \geq 1 \right\}$$

the margin constraint corresponding to the member $x$ from the multi-instance $X_i$. In order for the ensemble to satisfy a positively-labeled multi-instance soft margin constraint, the solution $(\alpha, \xi)$ must lie in the following disjunctive set determined by this input

$$(\alpha, \xi) \in \bigcup_{x \in X_i} H_i(x) . \quad (4.10)$$

To simplify our discussion below, we will separate the members of negative multi-instances, since they are not ambiguous. In other words, we treat the members as independent negatively-labeled individuals. Since each constraint was relevant in the original formulation (negative multi-instances are not ambiguous), the only effect this has is to decouple the slack variables. The reasoning is the same as in Chapter 3. We can therefore
represent all margin constraints using the disjunctive set notation, under the assumption that negatively-labeled multi-instances have only one member. Finally, we have the positivity constraints on the variables $(\alpha, \xi)$; in other words, the solution must lie in the positive quadrant, denoted $Q \equiv \{(\alpha, \xi) : \alpha, \xi \geq 0\}$.

We can now formulate the following disjunctive program:

$$
\text{DPBoost } \min_{\alpha, \xi} \sum_{k=1}^{n} \alpha_k + C \sum_{i=1}^{m} \xi_i \\
\text{s.t. } (\alpha, \xi) \in Q \cap \bigcap_{i=1}^{m} \bigcup_{x \in X_i} H_i(x).
$$

Notice that if $K_i \geq 2$ then the constraint imposed by $X_i$ is non-convex, since it is defined via a union of halfspaces. Figure 4.2 illustrates with a simple example why optimization over a non-convex set is difficult. However, for multi-instances with $K_i = 1$, the resulting constraints are the same as in Equation (4.5). In other words, the constraint reduces to a single halfspace $\bigcup_{x \in X_i} H_j(x) = H_j$. Since we will handle these two cases quite differently in the sequel, let us introduce index sets $A = \{i : K_i \geq 2\}$ and $B = \{i : K_i = 1\}$. Note that the index set $B$ contains all negatively labeled inputs $\{i : y_i = -1\}$, since we have assumed these have a single members.

By carefully analyzing the nature of the resulting optimization problem, and following the approach of [6, 31], we can derive a sequence of successively stronger LP relaxations that can be used to compute lower and upper bounds on the objective. Since it turns out that exploiting sparseness is a crucial aspect, we have focused on a linear programming formulation by generalizing the LPBoost algorithm [25, 45, 17]. We call the resulting method Disjunctive Programming Boosting (DPBoost).

4.2.2 Convexification

Since global optimization over non-convex sets is difficult, we consider relaxations of our problem. A suitable way to define a relaxation of the non-convex optimization problem is to replace the disjunctive set in Equation (4.10) by its convex hull. The resulting approximation to the original problem is convex, and moreover, can be expressed as a linear program. As such, we can compute the global minimum value efficiently. Furthermore, convexification has the effect of enlarging the feasible region. Since the solution may lie outside the original feasible region, it may not satisfy the multi-instance margin constraints. As a result, it may not generalize as well as the globally optimal solution of the original non-convex problem. The rationale in using this relaxation is that the resulting convex optimization problem is tractable.
Let \( \text{cl-conv}(S) \) denotes the closure of the convex hull of the limiting points of \( S \) define. As suggested, we replace each disjunctive set in Equation (4.10) with its convex hull, to obtain the resulting feasible region

\[
(\alpha, \xi) \in Q \cap \bigcap_{i \in A} \text{cl-conv} \left[ \bigcup_{x \in X_i} H_i(x) \right] \cap \bigcap_{j \in B} H_j.
\]

This region is convex because the intersection of convex regions is convex. Notice that for the unambiguous examples, indexed by \( j \in B \), there is no need to employ convexity; the convex hull of a halfspace is the halfspace itself.

### 4.2.3 Parallel Reductions

As shown in [6], a whole hierarchy of such relaxations can be built, using the fundamental fact that \( \text{cl-conv}(S) \cap \text{cl-conv}(T) \supseteq \text{cl-conv}(S \cap T) \). A tighter convex relaxation is obtained, if we intersect as many sets as possible before taking their convex hull. See Figure 4.3. Since repeated intersections of non-trivial disjunctive sets leads to a combinatorial blow-up in the number of constraints illustrate this fact, we propose to intersect each disjunctive multi-instance margin constraint with all unambiguous halfspace constraints, as well as with \( Q \). This is called a parallel reduction step [6]. It results in the following convex relaxation of the constraints in Equation (4.10)

\[
(\alpha, \xi) \in \bigcap_{i \in A} \text{cl-conv} \left[ \bigcup_{x \in X_i} \left( H_i(x) \cap Q \cap \bigcap_{j \in B} H_j \right) \right].
\]  

Given a solution to the relaxed problem, one can easily obtain lower and upper bounds on the solution value of the original problem. First, since the feasible region of the original problem is included in that of the relaxed version, its minimal value is a lower bound for the original problem \( H^*_\text{relaxed} \leq H^*_\text{boost} \). An upper bound \( H^*_\text{boost} \leq H^*_\text{projected} \) is obtained using the objective value after projecting the relaxed solution onto the original feasible space. The projection is done using the following heuristic iteratively. First, a witness member from one of the remaining ambiguous multi-instances is selected. All remaining member instances from the chosen multi-instance are removed from the problem. The revised feasible approximation yields a solution that is closer to the original feasible space. Upon convergence, all ambiguity is removed, and the solution therefore lies in the original feasible space. The objective value obtained at the point is an upper bound on the true minimum, by definition.
Figure 4.2: The geometry of DPboost. On the left, we display an example of the simplest possible feasible region, or version space (shaded) corresponding to a single ambiguous example with two member instances. The non-convexity of the version space is the result of the ambiguity. The dimensions correspond to variables $\alpha_k, \xi_i$, but we are only viewing a 2-dimensional projection. Notice that the version space is bounded positive orthant. A more realistic case would include more than one ambiguous example, and some unambiguous ones too, in which case there would be more than one non-convex constraint and additionally some halfspace constraints; the version space would be obtained by intersection. Despite the simplicity of this example, one can still see how “pockets” may prevent a descent based algorithm from finding the global minima of a linear objective over this region. The main idea proposed by the DPboost algorithm is to remove the “pockets”, by approximating the shaded region as shown on the right.

Figure 4.3: The geometry of DPboost (continued). The left figure displays the effect on the version space had by adding one additional unambiguous example and intersecting with its corresponding halfspace constraint, after convexification. The right figure displays the effect had before convexification. This is called a reduction. Notice the shrinking effect.
4.2.4 Linearization

There is a lift-and-project representation of the convex hulls in Equation (4.12), i.e., one can characterize the feasible set as a projection of a higher dimensional polyhedron which can be explicitly characterized, as shown in [6].

**Proposition 1.** Assume a set of non-empty linear constraints \( H_i \equiv \{ z : A^i z \geq b^i \} \neq \emptyset \) is given. Then \( z \in \text{cl-conv} \bigcup_i H_i \) if and only if there exist \( z^j \) and \( \eta^j \geq 0 \) such that

\[
    z = \sum_j z^j, \quad \sum_j \eta^j = 1, \quad A^j z^j \geq \eta^j b^j.
\]

**Proof.** [6]

Let us pause here briefly and recapitulate what we have achieved so far. We have derived a LP relaxation of the original disjunctive program for boosting with ambiguity. This relaxation was obtained by a linearization of the original non-convex constraints. Furthermore, we have demonstrated how this relaxation can be improved using parallel reduction steps.

Applying this linearization to every convex hull in Equation (4.12) individually, notice that one needs to introduce duplicates \( \alpha^x, \xi^x \) of the parameters \( \alpha \) and slack variables \( \xi \), for every \( x \in X_i \). In addition to the constraints \( \alpha^{x^k}_i, \xi^k_i, \xi^x_j, \eta^x_i \geq 0 \), the relevant constraint set for each individual multi-instance \( X_i, i \in A \) of the resulting LP can be written as

\[
\begin{align*}
    \forall x \in X_i : & \quad y_i \sum_k \alpha^{x^k}_i h_k(x) + \xi^x_i \geq \eta^x_i, \quad (4.13) \\
    \forall x \in X_i, \forall j \in B : & \quad y_j \sum_k \alpha^{x^k}_i h_k(x_j) + \xi^x_j \geq \eta^x_i, \quad (4.14) \\
    \forall j \in B : & \quad \xi_j = \sum_{x \in X_i} \xi^x_j, \quad (4.15) \\
    \forall k : & \quad \alpha_k = \sum_{x \in X_i} \alpha^{x^k}_i, \quad (4.16) \\
    \xi_i = & \sum_{x \in X_i} \xi^x_i, \quad (4.17) \\
    1 = & \sum_{x \in X_i} \eta^x_i \quad (4.18)
\end{align*}
\]

The first margin constraint in Equation (4.13) is the one associated with the specific instance \( x \), while the
second set of margin constraints in Equation (4.14) stems from the parallel reduction performed with unambiguous member examples.

### 4.2.5 Duality

We have just derived a relaxed version of the original ensemble learning problem that we proposed for learning from multi-instances $X_i$. Following the approach taken in LPBoost, we now calculate the dual of the above relaxation. The key result from this derivation is a complicated constraint in terms of the dual $u$-variables

$$\forall i, \forall x \in X_i : y_i u_i^x h_k(x) + \sum_{j \in B} y_j u_j^x h_k(x_j) \leq \rho_{ik}, \quad \sum_{i \in A} \rho_{ik} = 1. \quad (4.19)$$

Recall that we have defined the following primal variables $\alpha_k, \alpha_k^x, \xi_i, \xi_i^x, \xi_j^x$, and $\eta_i^x$. To write down the dual problem, we introduce dual variables $u^x$ and $u_j^x$ for the margin constraints, and $\rho_{ik}, \sigma_i$, and $\theta_i$ for the equality constraints on $\alpha_k, \xi$ and $\eta$, respectively. The Lagrangian is formed by adding each constraint from the primal problem to the objective, multiplied by the corresponding dual variable. We arrive at

$$L = \sum_k \alpha_k + C \left( \sum_i \xi_i + \sum_j \xi_j \right) - \sum_i \sum_{x \in X_i} u_i^x \left( y_i \sum_k \alpha_k^x h_k(x) + \xi_i^x - \eta_i^x \right)$$

$$- \sum_i \sum_{x \in X_i} \sum_j u_j^x \left( y_j \sum_k \alpha_k^x h_k(x_j) + \xi_j^x - \eta_j^x \right) + \sum_i \theta_i \left( 1 - \sum_{x \in X_i} \eta_i^x \right)$$

$$- \sum_{i,k} \rho_{ik} \left( \alpha_k - \sum_{x \in X_i} \alpha_k^x \right) - \sum_i \sigma_i \left( \xi_i - \sum_{x \in X_i} \xi_i^x \right) - \sum_{i,j} \sigma_{ij} \left( \xi_j - \sum_{x \in X_i} \xi_j^x \right)$$

$$- \sum_i \sum_{x \in X_i} \sum_k \tilde{\alpha}_k^x \alpha_k^x - \sum_i \sum_{x \in X_i} \tilde{\xi}_i^x \xi_i^x - \sum_i \sum_{x \in X_i} \sum_j \tilde{\xi}_j^x \xi_j^x - \sum_i \sum_{x \in X_i} \tilde{\eta}_i^x \eta_i^x.$$
Taking derivatives w.r.t. primal variables, and simplifying, leads to the following dual problem

\[
\text{DPboost-dual} \quad \begin{align*}
\max & \quad \sum_i \theta_i \\
\text{s.t.} & \quad \theta_i \leq u_i^x + \sum_j u_j^x, \\
& \quad u_i^x \leq C, \\
& \quad u_j^x \leq \sigma_{ij}, \\
& \quad \sum_i \sigma_{ij} \leq C, \\
& \quad y_i u_i^x h_k(x) + \sum_j y_j u_j^x h_k(x_j) \leq \rho_{ik}, \\
& \quad \sum_i \rho_{ik} \leq 1.
\end{align*}
\]

A more detailed analysis of these equations appears in Chapter 6.

### 4.2.6 Row and Column Selection

The size of the resulting problem is significant. To make this explicit, we first defining quantities \( q = \sum_{i \in A} K_i \) and \( r = \sum_{i \in B} K_i \) which denote the number of instances in ambiguous and unambiguous bags, plus the quantity \( p = |A| \) denoting the number of ambiguous bags. Then, counting the variables in the primal (columns), we note that there are \( O(q \cdot n) \) replicated \( \alpha_k^x \) variables, and \( O(q \cdot r) \) replicated \( \xi_j^x \) variables; and that both \( q \) and \( r \) are \( O(m) \). Thus, as a result of linearization and parallel reductions, the number of variables in the primal LP is now \( O(m \cdot n + m^2) \), compared to \( O(n + m) \) of the standard LPBoost. Similarly, counting dual variables (rows), we note that there are \( O(q \cdot r) \) margin constraints due to the parallel reductions, and \( O(p \cdot n) \) constraints due to the summation constraints on \( \alpha_k^x \). Thus, the number of constraints has also been inflated significantly from \( O(m) \) in LPBoost, to \( O(m^2 + m \cdot n) \).

In order to maintain the spirit of LPBoost in dealing efficiently with a large-scale linear program, we propose to maintain the column selection scheme of selecting one or more \( \alpha_k^x \) in every round. Notice that all \( \alpha_k^x \) variables are linked through the equality constraints \( \sum_{x \in X_i} \alpha_k^x = \alpha_k \) for all \( X_i \). Thus, the column selection can not proceed for every multi-instance \( X_i \) independently. In particular, \( \alpha_k^x > 0 \) implies \( \alpha_k > 0 \), so that \( \alpha_k^w > 0 \) for at least some \( w \in X_i \) for each \( i \in A \). We hence propose to simultaneously add all columns \( \{ \alpha_k^x : x \in X_i, \ i \in A \} \) involving the same weak hypothesis and to prune those back after each boosting round in order to exploit the expected sparseness of the solution. In order to select a feature \( h_k \), we compute the
CHAPTER 4. LEARNING FROM AMBIGUOUS EXAMPLES WITH LINEAR PROGRAMMING BOOSTING

The following score

\[ S(k) = \sum_{i} \bar{\rho}_{ik} - 1, \quad \bar{\rho}_{ik} \equiv \max_{x \in X_i} \left[ y_i u_i^x h_k(x) + \sum_{j \in B} y_j u_j^x h_k(x_j) \right]. \] (4.20)

Notice that due to the block structure of the tableau, working with a reduced set of columns also eliminates a large number of inequalities (rows). However, the large set of \( q \cdot r \) inequalities for the parallel reductions is still prohibitive.

In order to address this problem, we propose to perform incremental row selection in an outer loop. Once we have converged to a column basis for the current relaxed LP, we add a subset of rows corresponding to the most useful parallel reductions. One can use the magnitude of the margin violation as a heuristic to perform this row selection. Hence we propose to use the following score

\[ T(x, j) = \eta_i^x - y_j \sum_k \alpha_k^x h_k(x_j), \quad \text{where } x \in X_i, \ i \in A, \ j \in B \] (4.21)

This means that for current values of the duplicated ensemble weights \( \alpha_k^x \), one selects the parallel reduction margin constraint associated with ambiguous member instance \( x \) and unambiguous instance \( j \) that is violated most strongly.

Although the margin constraints imposed by unambiguous training instances \( (x_j, y_j) \) are redundant after we perform parallel reduction steps in Equation (4.12), we still add them to the problem. Since these constraints are unambiguous, the features added during the first iteration will therefore satisfy at least some constraints that have been derived directly from the training examples. We expect this to give us a better starting point with respect to the row selection process. Moreover, if time or memory is limited, the features selected before row selection converges, or even those selected before row selection begins, will be meaningful. We hence add the following constraints to the primal

\[ y_j \sum_k \alpha_k h_k(x_j) + \xi_j \geq 1, \quad \forall j \in J, \] (4.22)

which will introduce additional dual variables \( u_j, j \in J \). Notice that in the worst case where all inequalities imposed by training multi-instances \( X_i \) are vacuous, this will make sure that one recovers the standard LPBoost formulation on the individual member instances. One can then think of the row generation process as a way of deriving useful information from ambiguous examples. As rows are added to the approximation,
the feasible regions corresponding to ambiguous candidate member instances are eliminated from consider-
ation if they are not consistent with the unambiguous constraints. This information takes the form of linear
inequalities in the high dimensional representation of the convex hull and will sequentially reduce the version
space, i.e. the set of feasible \((\alpha, \xi)\) pairs.

The difference \(H^*_{\text{projected}} - H^*_{\text{relaxed}}\) can be used as a termination condition during learning. It signals
when we have already found an optimal solution to the original disjunctive programming problem. If needed,
the upper and lower bounds can be further strengthened by using them in combination with branch-and-bound
search.

More analysis of this algorithm, including a demonstration of the upper and lower bounds mentioned
above, will appear in the thesis.

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{input}: training sample \(D\)
\State initialize \(H = \emptyset\), \(C = \{\xi_i : i \in A \cup B\}\), \(R = \{u^*_x : x \in X_i, i \in A\} \cup \{u^*_j : j \in B\}\)
\State \(u^*_j = \frac{1}{|J|}\), \(u^*_x = 0\), \(\xi_i = 0\)
\Repeat
\State \text{column selection}: select \(h_k \notin H\) with maximal \(S(k)\)
\State \(H = H \cup \{h_k\}\)
\State \(C = C \cup \{\alpha_k\} \cup \{\alpha^*_x : \forall x \in X_i, \forall i \in A\}\)
\State solve \(\text{LP}(C, R)\)
\State \textbf{until} \(\max_k S(k) < \epsilon\)
\State \text{row selection}: select a set \(S\) of pairs \((x, j) \notin R\) with maximal \(T(x, j) > 0\)
\State \(R = R \cup \{u^*_x : (x, j) \in S\}\)
\State \(C = C \cup \{\xi^*_x : (x, j) \in S\}\)
\State solve \(\text{LP}(C, R)\)
\State \textbf{until} \(\max_{x, j} T(x, j) < \epsilon\)
\end{algorithmic}
\end{algorithm}

**4.3 Experiments**

(*More explanation and editing needs to be done in this section.*)

**4.3.1 Proof of Concept**

We generated a set of synthetic weakly labeled data sets to evaluate DPBoost on a small scale. These were
multiple-instance data sets, where the label uncertainty was asymmetric; the only ambiguous bags \(|X_i| > 1\)
were positive. More specifically, we generated instances \(x \in [0,1] \times [0,1]\) sampled uniformly at random
from the white \((y_i = 1)\) and black \((y_i = -1)\) regions of Figure 4.4, leaving the intermediate gray area as
a separating margin. The degree of ambiguity was controlled by generating ambiguous bags of size $k \sim \text{Poisson}(\lambda)$ having only one positive and $k - 1$ negative instances. To control data set size, we generated a pre-specified number of ambiguous bags, and the same number of singleton unambiguous bags.

As a proof of concept benchmark, we compared the classification performance of DPBoost with three other LPBoost variants: perfect-knowledge, perfect-selector, and naive algorithms. All variants use LPBoost as their base algorithm and have slightly different preprocessing steps to accommodate the MIL data sets. The first corresponds to the supervised LPBoost algorithm; i.e. the true member instance labels are used. Since this algorithm does not have to deal with ambiguity, it will perform better than DPBoost. The second algorithm uses the true member instance labels to prune negative instances from ambiguous bags and thus solves a smaller and fully supervised problem using LPBoost. This algorithm provides an interesting benchmark, since its performance is the best we can hope for from DPBoost. At the other extreme, the third variant assumes the ambiguous member instance labels are equal to their respective bag labels. For all algorithms, we used thresholded “RBF-like” features.

Figure 4.5 shows the discriminant boundary (black line), learned by each of the four algorithms for a data set generated with $\lambda = 3$ and having 20 ambiguous bags (i.e. $|I| = 20$, no. ambig. = 71, no. total = 91). The ambiguous instances are marked by “o”, unambiguous ones “x”, and the background is shaded to indicate the value of the ensemble $F(x)$ (clamped to $[-3, 3]$). It is clear from the shading that the ensemble has a small number of active features for DPBoost, perfect-selector and perfect-knowledge algorithms. For each classifier, we report the classification accuracy on individual instances sampled uniformly on a (21 x 21) grid. The sparsity of the dual variables was also verified; less than 20 percent of the dual variables and reductions were active.

We ran 5-fold cross-validation on the synthetic data sets for $\lambda = 1, 3, 5, 7$ and for data sets having $|I| =$
Figure 4.5: Discriminant boundaries learned by naive (accuracy = 53.3 %), DPBoost (85.3 %), DPBoost with projection (85.3 %), perfect-selector (86.6 %) and perfect-knowledge (92.7 %) algorithms.

10, 20, 30. Figure 4.4 (right side) shows the mean classification accuracy for individual instances with error bars showing one standard deviation, as a function of the parameter $\lambda$.

### 4.4 Discussion

This chapter presents a novel learning algorithm for classification problems where labels are associated with multi-instances instead of individuals. Using synthetic data, the expected behaviour of the algorithm has been demonstrated. Our current implementation could not handle large data sets, and so improvements, followed by a large-scale validation and comparison to other algorithms using benchmark MIL data sets, will follow.

A new formulation, which proposes a new type of column selection, is proposed in Section 6.
Chapter 5

Learning from Ambiguous Examples for Object Recognition

This chapter discusses an application of learning from ambiguous examples to the domain of object recognition. In this domain, polymorphic representations of the input are obtained by detecting “interest points” within each image. Each interest point, and the surrounding region is a potential “part” of an object. An object is detected with reasonable accuracy when a group of parts are detected in a stable configuration.

This chapter also introduces a novel algorithm for dealing with ambiguity. The new algorithm has several distinguishing characteristics. First of all, it operates in rounds in a boosting framework. The result is an ensemble of weak learners that operate as detectors. Each weak learner uses the principle of consistent disambiguation to select discriminative interest points in the training images. These disambiguations made by weak learners need not correspond in consecutive boosting rounds. Because the weak learners operate independently, the ensemble naturally acquires diversity and redundancy that is critical for object recognition.

Secondly, the weak learners consider new label semantics that are appropriate for recognizing objects. In particular, instead of using the 1-of-$N$ semantics, the weak learners consider multi-instance labels that arise from conjunctions of one or more member instance labels. Not only does the weak learner discover the underlying concepts for the terms of a conjunction, it automatically selects the number of terms in a conjunction to best predict the overall label. In the terminology of the application, the weak learners select collections of geometrically stable parts that are useful for recognition.
5.1 Introduction

Given an image and a target object category, the object detection task is to localize an instance of the target object, or decide none exist. The problem is difficult due to the complex variabilities of objects due to position, pose, and deformations of the object, not to mention lighting effects, such as reflections and specularities, and finally due to issues of occlusion or spurious duplication, or false positive detection, of features.

Learning a wide variety of object categories requires a system that can adapt its underlying representation to unexpected objects and imaging conditions. We employ a feature selection framework to guide the construction of object specific classifier, using a novel and descriptive family of image features. This work builds on 1) the constellation model for representing objects [19], 2) linear programming boosting for feature selection [17], 3) modeling with permutations [27], and 4) disjunctive programming optimization [6]. We experiment with labeled data, and briefly discuss unlabeled data scenarios.

We define a family of constellation feature functions which detect geometrically stable groups of parts. Each such function is defined by the appearance and shape parameters of a dedicated collection of parts: a constellation. The value of a constellation feature function, when applied to an image, is a measure of similarity reflecting the group of image parts that is most similar to the constellation. By virtue of selecting the maximally similar group of image parts, the feature is non-linear. Efficient matching methods are used to detect the most similar instance of a given constellation. The locations of parts found by constellation features may be used to localize the object in the image.

We use a boosting algorithm to construct an ensemble of constellation features. Variety in the selected features naturally account for part occlusions and significant variations in pose or lighting. Boosting is formulated as a convex optimization problem. In each boosting round, the goal of the weak learner is to select the constellation feature function that is most likely to improve the ensemble. For tractability, we employ a greedy algorithm and heuristic optimization techniques to select the underlying constellation one part at a time. While this solution is sub-optimal, initial results are promising. Even still, the single-part learning problem is difficult. The input examples are ambiguous. The location of the part in the image must be inferred. The techniques for learning from ambiguous examples are adopted for this task. For this reason, we call the method “Boosted Ambiguous Feature Learning”.

The convex learning framework that we use for boosting is based on empirical risk minimization. The algorithm can be used for classification or density estimation. Notably, there are no probabilistic modeling
assumptions made about constellation parts or their compositions. To facilitate matching and learning, we filter the input image for candidate parts using uniform sampling, or keypoint detectors. Thus, we consider the input image to be a collection \( X = \{x_1, \ldots, x_p\} \) of local image regions.

5.2 Constellation Features

A constellation model is specified by the appearance of and the pairwise geometric relationships among its parts. Being flexible, a constellation may assume a large number of configurations. We consider all configurations which arise by matching keypoints in the image to parts of the constellation. The sum of the respective appearance and relative geometry similarities, between image keypoints and the constellation model, determines an overall match quality.

Constellation part appearances are specified by the matrix \( W^a = \{a(w_1), \ldots, a(w_r)\} \) where each column \( a(w_r) \in \mathbb{R}^d \) is a local region descriptor such as a vector of pixel values, or SIFT features [33]. An inner product \( \langle a(w), a(x) \rangle \) determines the appearance similarity of constellation part \( w \) and image part \( x \).

Inter-part geometric relationships are represented by the matrix \( W^g \in \mathbb{R}^{r \times r} \) whose entries \( d(w_i, w_j) \) are defined by a function \( d \) such as the distance between parts. Other relations may depend on the relative scale or orientation of parts. The geometric similarity between a pair of constellation parts \( w_1, w_2 \) and a pair of image parts \( x_1, x_2 \) is given by \( \langle d(w_1, w_2) \cdot d(x_1, x_2) \rangle \). For simplicity of notation only, we assume there is only one geometric relation \( d \) in the proceeding development.

In an analogous fashion, we define a matrix \( \Phi^a = \{a(x_1), \ldots, a(x_p)\} \), for the appearances of candidate parts, and a matrix \( \Phi^g \in \mathbb{R}^{p \times p} \) with entries \( d(x_i, x_j) \), for the inter-part geometric relationships among candidates, from an image \( X \).

To express for the overall match quality, we introduce the truncated permutation matrix \( P \in \Pi^{r \times p} \), which encodes a match of image keypoints to constellation parts. A truncated permutation matrix has entries \( P_{ij} \in \{0, 1\} \) that satisfy the following row and column constraints: \( \sum_j P_{ij} \leq 1 \) for \( 1 \leq i \leq r \), and \( \sum_i P_{ij} = 1 \) for \( 1 \leq j \leq p \). For a \( p \times p \) matrix, left multiplication by \( P \) reorders rows and truncates columns, while right multiplication by \( P^T \) reorders columns and truncates rows. For example \( \Phi^a P^T \) is a \( d \times r \) matrix (as is \( W^a \)). Similarly, \( P \Phi^g P^T \) is a \( r \times r \) matrix (as is \( W^g \)). Finally, defining \( \langle A, B \rangle \equiv \sum_{ij} A_{ij} B_{ij} \) for matrices \( A \) and \( B \) of the same size, we see that the overall match quality is

\[
Z(X, W, P) = \langle W^g, P \Phi^g P^T \rangle + \langle W^a, \Phi^a P^T \rangle. \tag{5.1}
\]
CHAPTER 5. LEARNING FROM AMBIGUOUS EXAMPLES FOR OBJECT RECOGNITION

The infinite family $\mathcal{F}$ of constellation feature functions using the definitions above. Each feature function $f \in \mathcal{F}$ is parameterized by a constellation model $W \equiv (W^a, W^g)$. For reasons that will become clear later, we place a bound on the norm of the parameters $\|W\|_1 \leq 1$. Constellation feature functions $f : X \rightarrow \mathbb{R}$ operate by first searching for the best configuration in the image $X$, and then returning the resulting match quality. Using the expression from Equation (5.1), the constellation feature function may be expressed as an integer quadratic programming problem over the variable $P$

$$f (X; W) = \max_{P \in \Pi} Z (X, W, P).$$

This NP-hard combinatorial optimization problem is an instance of the well-studied quadratic assignment problem (QAP), for which many efficient approximations are known [40, 4].

5.2.1 LPQAP Approximation

Computing the best configuration of a constellation in an image is equivalent to solving a quadratic assignment problem (QAP) over $P$. Writing $P = (p_{ij})$, we have $\sum_j p_{ij} = 1, \forall i$, $\sum_i p_{ij} = 1, \forall j$ and $p_{ij} \in \{0, 1\}$. It is quadratic due to the interactions between parts, which are indexed by the products $p_{ij}p_{kl}$. Expanding Equation (5.1), the quadratic assignment can be written as

$$\text{QAP} \quad \max_u \sum_{i,j,k,l} d_{ijkl} p_{ij} p_{kl}$$

subject to $P \in \Pi$. (5.4)

We will make use of the LPQAP relaxation. This approach involves lifting via the implicit definition of new variables $q_{ijkl} = p_{ij}p_{kl}$; and relaxing the integrality conditions, i.e. $p_{ij} \in [0, 1]$. This results in the following LP

$$\text{LPQAP} \quad \min_q \sum_{i,j,k,l} d_{ijkl} q_{ijkl}$$

subject to $\sum_l q_{ijkl} = p_{ij}, \forall i, j, k$, $\sum_k q_{ijkl} = p_{ij}, \forall i, j, l$ (5.6)

$\sum_j p_{ij} = 1, \forall i$, $\sum_i p_{ij} = 1, \forall j$ (5.7)

$q_{ijkl} = q_{klij}, \forall i, j, k, l$ (5.8)

$p_{ij} \geq 0, q_{ijkl} \geq 0, \forall i, j, k, l$. (5.9)
While we could simply relax the quadratic problem and solve with a quadratic solver, the lifting stage is beneficial in two ways. First, the problem is linear and the dual program is solved efficiently using variants of the Hungarian method [40, 4]. Second, by relaxing integrality on the lifted $q_{ijkl}$ variables, the effective growth of the feasible region in the original space is reduced [40, 4].

The solution $P^*$, which is a member of the closed convex hull of truncated permutation matrices $\text{cl-conv}(\Pi)$, is a row-stochastic matrix; in other words, it is possible that $P^*$ represents a distribution over several favored configurations, although this rarely happens in our setting. While there exist techniques for improving the approximation [34, 10], we prefer this approximation because it captures the appearance and geometry of the constellation, and is very efficient. This is essential for the weak learner, which repeatedly evaluates the current $f$.

### 5.3 Selecting Constellation Features

The goal of feature selection is to pick an informative collection of constellation features from $\mathcal{F}$. This could be for a supervised task, such as learning a classifier, or an unsupervised task, such as 1-class density support learning. In the latter case, the selected features may reveal important intra-class structures. We assume a collection of training examples $\mathcal{D} = \{(X_i, y_i), 1 \leq i \leq m\}$. Labels $y_i$ are $\{-1, 1\}$. For the unsupervised setting, labels are assigned uniformly $y_i = 1$, $1 \leq i \leq m$.

Because the framework of linear programming boosting (LPBoost) [17] has previously been applied to both 1-class and 2-class problems, we use it to guide the selection of features. The LPBoost formulation also leads to sparse solutions. Other boosting algorithms are applicable in the classification setting. LPBoost incrementally constructs an ensemble of the following form

$$F(X) = \sum_{k=1}^{N} \alpha_k f_k(X).$$ (5.10)

Upon convergence, the chosen features define a mapping of the inputs $X \rightarrow (f_1(X), \ldots, f_T(X))$. The weights $\alpha_k$ define a hyperplane that is used for classification or outlier detection by thresholding $\hat{y} = \text{sgn} F(X)$. Positive values indicate that the target is present (classification), or that the input is not an outlier (1-class).
5.3.1 LPBoost Revisited

As shown in [17], a maximum-margin approach for learning ensemble coefficients can be formulating the following linear program

\[
\text{LPboost} \quad \min_{\alpha, \xi} \sum_{k=1}^{n} \alpha_k + C \sum_{i=1}^{m} \xi_i \quad \text{(5.11)}
\]

\[
\text{s.t.} \quad y_i \sum_{k=1}^{n} \alpha_k f_k(X_i) \geq 1 - \xi_i, \quad \forall i \quad \text{(5.12)}
\]

\[
\alpha_k \geq 0, \quad \xi_i \geq 0. \quad \text{(5.13)}
\]

The LPBoost formulation is based on the minimization of the hinge loss, instead of the exponential loss that is used in Adaboost. The linear programming formulation is convex and thus amenable to global optimization.

In our case, the weak hypotheses are constellation features \( f_k(\cdot) \equiv f(\cdot; W_k) \). Since there are an infinite number of potential constellation features, the constraint matrix for the LP clearly does not fit in memory. LPBoost works in rounds, solving restricted sub-problems of increasing size until it has found the true solution to the unrestricted problem, using a finite collection of columns. The algorithm maintains a dual variable \( u_i \geq 0 \) for each training instance, called the \textit{misclassification cost}. Initially, no features participate in the ensemble; all coefficients \( \alpha_k \) are zero, and the misclassification costs are uniform. Constellation features are added to the ensemble when they are most likely to move the solution to a lower objective value, as determined by the amount the feature violates its respective constraint in the dual LP. Thus, at round \( t \), the constellation features having the largest dual constraint violation is found by maximizing

\[
\text{CG} \quad \max_{W} \sum_{i=1}^{m} u_i y_i f(X_i; W) = \max_{W} S(f(\cdot; W)). \quad \text{(5.14)}
\]

\[
\text{s.t.} \quad \|W\|_1 \leq 1. \quad \text{(5.15)}
\]

This is known as the column generation problem. The objective quantifies the discriminative power of \( f \) on the weighted instances, and is thus known as the constellation feature score. Since only the sign and relative values of features are of importance, we have imposed a bound on the constellation parameters, and indirectly bounded the objective function in Equation (5.14). The solution determines the parameters \( W \) of the chosen constellation feature. After selection, LPBoost reoptimizes the weights \( \alpha_k \) and misclassification costs \( u_i \). When the optimal score is less than 1, then all dual constraints have been satisfied. Other insights into the sparse solution structure can be inferred from the conditions of complementarity as discussed in [17].
Algorithm 3 LPBoost

1: **input:** training sample $D$
2: Initialize $H = \emptyset$, $\alpha = \emptyset$, $u = (u_1, \ldots, u_m)$, $u_i = \frac{1}{|\mathcal{M}|}$, $\forall i$, $k = 0$
3: **repeat**
4: $k = k + 1$, Solve $CG(D, u)$ for $f_k = f(\cdot; W_k)$
5: $H = H \cup \{f_k\}$, $\alpha = \alpha \cup \{\alpha_k\}$
6: Solve $LP(D, H)$ for $\alpha$ and $u$
7: **until** $S(f_k) < 1 + \epsilon$

5.3.2 Case 1: Single Part Constellation Features

For the moment, we consider constellations having only one part. In this case, $P \in \Pi$ is a simple row vector, there are no geometric features $W^g$, and feature selection results in the following non-linear optimization problem for $W$

\[
CG \quad \max_W \sum_{i=1}^m u_i y_i \max_{P \in \Pi} \langle W^a, \Phi^a, P^T \rangle \quad (5.16)
\]
\[
\text{s.t.} \quad \|W\|_1 \leq 1. \quad (5.17)
\]

We can express the single-part CG problem as a disjunctive program, through the introduction variables $\rho_i$ for each training example

\[
CGDP \quad \max_{W, \rho} \sum_{i=1}^m u_i y_i \rho_i \quad (5.18)
\]
\[
\text{s.t.} \quad \bigvee_{P \in \Pi} (\rho_i \leq \langle W^a, \Phi^a, P^T \rangle), \; \forall i \quad (5.19)
\]
\[
\bigwedge_{P' \in \Pi} \left(\langle W^a, \Phi^a, P'^T \rangle \leq \rho_i \right), \; \forall i \quad (5.20)
\]
\[
\|W\|_1 \leq 1. \quad (5.21)
\]

Once in this form, we can apply the theory of disjunctive programming to derive an efficient approximation. As long as it achieves a score $S(f(\cdot; W)) > 1 + \epsilon$, the resulting single-part constellation feature has the potential to decrease the learning objective in Equation (5.11).
5.3.3 Disjunctive Programming Approximation

The single-part CGDP optimization problem in Equation (5.18) is challenging since the feasible region is not convex; it is the union of halfspaces defined via Equations 5.19-5.21. We appeal to disjunctive programming techniques first proposed in [6] and adopted in [1]. Disjunctive programming obtains a linear approximation by loosening the disjunctive constraints. In particular, letting $S_i$ denote the region of feasibility corresponding to the $i$-th disjunctive constraint from Equation (5.19), then the relaxation substitutes a linear representation of $cl-conv(S_i)$ in place of $S_i$. Making the following substitutions

variables: $w = W^a$ (5.22)
coefficients: $x_P^i = \Phi^a x_P^T$ (5.23)

allows us to write the original region of feasibility as

$$S_i = \bigcup_{P \in \Pi} \{ w : (\rho_i \leq \langle x_P^i, w \rangle) \land (\|w\|_1 \leq 1) \}.$$ (5.24)

Through the introduction of variables $w_P^i, \eta_P^i, \rho_P^i (\forall P \in \Pi)$, a lifting technique is used to represent $cl-conv(S_i)$, resulting in the following set of linear constraints

$$\forall P \in \Pi : \|w_P^i\|_1 \leq \eta_P^i$$ (5.25)
$$\forall P \in \Pi : \rho_P^i \leq \langle w_P^i, x_P^i \rangle$$ (5.26)
$$w = \sum_{P \in \Pi} w_P^i, \; \rho_i = \sum_{P \in \Pi} \rho_P^i, \; 1 = \sum_{P \in \Pi} \eta_P^i$$ (5.27)

This application of disjunctive programming differs from that of [1] in that the problem size is greatly restricted due to (1) the sparsity of $u_i$ variables, and (2) the low dimensional feature space $x \in \mathbb{R}^d$. As in [1], we obtain better results by further constraining each $S_i$ before taking the convex hull. For this purpose, we fold all conjunctive constraints from Equation (5.20) into the definition of $S_i, \forall i$.

5.3.4 Case 2: Multi-Part Constellation Features

The above strategy cannot be applied for multi-part constellation features, due to the exponential growth in the number of valid truncated permutations $P \in \Pi^{r \times p}$, as $r$ increases. For this reason, and because it is sufficient to find a constellation feature whose score is greater than 1, we adopt a greedy strategy whereby
constellation parameters are selected in rounds. Inspiration for this approach comes from [55, 49], where similar combinatorial selection problems are encountered.

The idea is to continue growing the constellation as long as the constellation score $S(f(\cdot; W))$ can be improved. In the $r$-th round, the goal is to select the appearance of the $r$-th part, as well as the inter-part geometry relating the $r$-th part to parts 1 through $r - 1$ of the constellation. These parameters are selected via a reduction to the single-part CGDP problem.

At the beginning of each stage, assume image keypoints $x_{p(1)}^{(1)}, \ldots, x_{p(r-1)}^{(r-1)}$ have been matched to the current constellation. These parts are removed from the image. Each remaining candidate image part $x$ is used to create a pseudo-part whose appearance $\bar{a}(x)$ contains the appearance of $x$, as well as the inter-part geometry relating $x$ to $x_{p(1)}, \ldots, x_{p(r-1)}$. Next, the single-part CGDP problem from Section 5.3.2 is solved with the pseudo-part training data $\bar{D}$. The solution is deconstructed, yielding the new constellation part appearance and inter-part geometry parameters.

Finally, at the end of each iteration, the augmented constellation feature function is evaluated on all training examples by solving the LPQAP approximation, and the stochastic truncated permutation matrix $P^*_{\tau}$ is used to identify the specific image keypoints matched to constellation parts; random sampling is used if the entries of $P^*_{\tau}$ are fractional. Failure to increase the overall constellation score $S(f(\cdot; W))$ naturally terminates constellation growth. Details are given in the algorithm below.

**Algorithm 4** Greedy constellation growth.

1. **input**: Training sample $D$ and misclassification costs $u = (u_1, \ldots, u_m)$
2. Initialize $D = D$, $W^a = [], W^g = [], r = 0$, $S^0 = 0$
3. repeat
4. $r = r + 1$
5. Solve CGDP ($D$, $u$) for parameter $w^*_r$
6. Deconstruct solution $\{a^T_r, d_{r,1}, \ldots, d_{r,r-1}, d_1, r, \ldots, d_{r-1}, r\} \leftarrow w^*_r$
7. for all $X_i$
8. Solve LPQAP ($W, X_i$) for $P^*$.
9. Project $P^* \leftarrow P^*$ where $P^* \in \Pi^{r \times p}$, and let $p = \text{diag}(1, \ldots, p) \bar{P}^* T$
10. Construct pseudo-appearances for $x_j \in X_i \setminus \{x_{p(1)}, \ldots, x_{p(r)}\}$
11. $\bar{a}_{ij} \leftarrow \{a^T_j, d_{j, p(1)}, \ldots, d_{j, p(r)}, d_{p(1), j}, \ldots, d_{p(r), j}\}$
12. end for
13. Compute objective $S^r = S(f(\cdot; W))$
14. until Number of parts exceeds threshold, or $S^r - S^{r-1} \leq 0$. 


Table 5.1: Shape improves recognition. Comparison of algorithm with and without the addition of inter-part geometry features, for different degrees of pixel noise. Mean accuracy and standard deviation are computed by 5-fold cross validation where, in each fold, the training set consisted of 100 sequences, and test set consisted of 25 sequences. Parameters used to generate the data: $\min_{fg} = 5$, $\max_{fg} = 5$, $N_{bg} = 10$, $L_{part} = 10$, $L_{seq} = 100$, $\lambda = 1$. Constellation features were restricted to have at most two parts.

5.4 Experiments

Our first experiment was highly controlled, relying on synthetically generated data. Images are one dimensional binary $\{-1, +1\}$ sequences of length $L_{seq} = N_{bg}L_{part}$. Background sequences have random i.i.d. Bernoulli (0.5) pixels. Target sequences contain a randomly perturbed instance of a multi-part template model, superimposed over the background.

First, the template model is generated randomly. A pre-specified number $\max_{fg}$ of template model parts of length $L_{part}$ are generated with i.i.d. Bernoulli (0.5) pixels, and respective locations for the template parts are randomly sampled from the discrete set $\{1, \ldots, N_{bg}\}$ without replacement. Next, the template model is used to generate foreground sequences. An integer $N_{fg}$, representing the number of observed foreground parts, is generated uniformly from the set $\{\min_{fg}, \ldots, \max_{fg}\}$. The identity of each observed part is drawn uniformly from the set of template parts $\{1, \ldots, \max_{fg}\}$ with replacement. Finally, we perturb part appearances and locations. For appearance variations, we add i.i.d. Bernoulli ($p$) pixel noise; while for spatial variations, we add integer offsets drawn i.i.d. Poisson ($\lambda$). We also include a common offset, again Poisson ($\lambda$), to the locations of all parts. With probability 0.5, any of the aforementioned location offsets are negated. Parts whose perturbed locations fall outside the range $\{1, \ldots, N_{bg}\}$ are ignored. If two parts have the same perturbed location, one is ignored. The remaining parts are superimposed over a random background sequence.

We compared two versions of the proposed algorithm. In the first [without shape] we ignored the inter-part geometry, while the second [with shape] used two inter-part geometry features $d_{1ij} = d_{ij}$ and $d_{2ij} = (N_{bg} - d_{ij})$, where $d_{ij}$ is the absolute distance between parts. It is encouraging to see that the latter variant performs better.
5.5 Discussion

This is work in progress. It is encouraging to see that the geometry can be used to improve recognition. The challenge will be to scale the algorithm up to real images where there are many more 10 candidate parts.
Chapter 6

Conclusions and Future Work

6.1 Learning from ambiguous examples with SVM

The SVM derives a classifier that characterizes a set of training examples by solving a large quadratic programming problem. Because the problem is convex, there is a unique solution. Theoretical results show that the classifier generalizes. Empirical results concur. Moreover, the algorithm scales well and is easy to use.

The extension of SVM to deal with ambiguous examples is conceptually simple. The notion of consistency is clear. The notion of disambiguation is clear. In this context, the natural (direct) expression of the combined disambiguation and learning problem results in a NP-hard mixed-integer quadratic programming problem. A two-stage local search algorithm was applied to the problem. Despite the theoretical challenge posed by the optimization problem, competitive results on benchmark data sets were obtained.

It is not known whether there is an algorithm that can find the global optimum efficiently. Following the approach used in the extension of LPBoost, the formulation of learning from ambiguous examples with SVM’s could be altered using concepts from disjunctive programming. This project would require careful attention to the special structure of active and inactive constraints, and is not for the faint-hearted. Therefore, this will be left as an open problem.

6.2 Learning from ambiguous examples with LPBoost

LPBoost derives a classifier that characterizes a set of training examples by solving a linear programming problem. Because the problem is convex, there is a unique solution. Theoretical results show that the classifier
generalizes. Empirical results concur. Due to the improved reliability/useability of packages for solving large-scale linear programs, the algorithm is scalable.

The extension of LPBoost to deal with ambiguous examples is conceptually simple. Initially, the approach is similar to that taken in the extension of SVM’s. The notion of consistency is clear. The notion of disambiguation is clear. In this case, the natural (direct) expression of the combined disambiguation and learning problem results in a NP-hard mixed-integer linear programming problem. Indeed, like the SVM extension, an analogous two-stage optimization procedure could have been devised.

Instead, methods of disjunctive programming are employed to derive an approximation to the original problem that is a linear programming problem. This LP has a number of rows and a number of columns that is quadratic in $D$, as compared to the number of rows and number of columns that is linear in $D$ for LPBoost. $D$ is the maximum of the number of features and the number of examples in the training data set. As an LP problem this approximation is convex.

A specialized algorithm that uses both row and column generation was proposed to solve the resulting problem. Empirical results on a set of carefully controlled test problems demonstrate the promise of this approach.

There were several issues with the initial implementation. These are:

1. Problematic choice of features. In the initial implementation, decision-stump features (weak hypotheses) were used. Decision stumps are defined for several thresholds of each feature. One problem with decision stumps, is that there are so many of them. A second problem is that they perform poorly in the 1-class context.

2. Inefficient use of CPLEX from Matlab through a MEX-based interface. This was a limitation of the system design. The LP was represented in sparse format in both Matlab and CPLEX. Management of the two sparse representations, and communication between Matlab and CPLEX was obstructed by the MEX-based interface. One limitation of the MEX-based interface was its simplicity. It provided rudimentary functionality for solving simple LP’s. In particular, the MEX-based interface could not utilize several features of CPLEX that were specifically designed for column generation. Furthermore, making changes to the MEX-based interface was laborious. For these reasons, the complete sparse representation in CPLEX was re-initialized from scratch on each call. Row and column generation heuristics were implemented in Matlab. The current set of rows and columns were communicated to CPLEX together with a starting basis. After the LP was solved by CPLEX, the solution was returned
to matlab and the memory allocated by CPLEX was cleared.

3. Conflicting steps of row and column generation. Recall that DPBoost employed alternating stages of row and column generation to define a sequence of approximations to the convex DPBoost formulation. The goal was to converge toward the overall solution without ever having to initialize and solve the entire problem. The row and column generation heuristics that were used determined the particular sequence of approximations. Column generation corresponded to the selection of additional features. Row generation corresponded to the selection of additional parallel reduction constraints. After converging to a column basis, new reductions were added to improve the approximation to the DPBoost formulation. While this approach was well motivated, it is not recommended in practice due to the opposing effects of row and column generation [7]. Upon adding additional constraints (rows) to the problem, the solution basis of the LP will typically change a great deal. This is not an issue when the sequence of problems is obtained by adding only columns or only rows.

The following is a list of ideas that we propose to address these problems above

1. There are two new types of features that are expected to improve performance.

   The first set of features to try are simply the individual, normalized features from feature space. The advantage of using these features is efficiency, because the number of these features is \( d \) while the number of decision stumps is typically \( dm \), where \( d \) is the dimension of the input feature space and \( m \) is the number of examples in the training set. Column generation will be quicker with fewer features. The second set of features that are expected to improve performance are derived from the decision-stump features that have been used thusfar. The modification involves weighting the features in a manner analogous to the term frequency - inverse document frequency (tf-idf) weighting that is commonly used to represent documents.

2. The efficiency of DPBoost can be improved dramatically by using the JAVA-based interface to CPLEX.

   This will make the algorithm efficient in two ways. First, using functionality available only to the JAVA API, CPLEX can be instructed to instantiate and maintain the LP throughout the DPBoost iterations. There is no need to store a duplicate copy of the constraint matrix in the memory allocated by Matlab. Secondly, the column generation optimizations of CPLEX may be used. These practical benefits were validated in the implementations of algorithms proposed in Chapter 5.
3. A new heuristic for the generation of columns can be used to generate a new sequence of LP approximations.

There are several reasons motivating the definition of this new approach. In our previous algorithm, all member instances participated in the approximation, and column generation was performed on the features. This approach is based on the selection of member instances within each multi-instance that are allowed to participate in the approximation. For this reason, it is a natural extension of the mixed-integer formulation (which one allows the witness to participate) that was used in Chapter 3 and that may have been applied equally well to LPBoost. Disjunctive programming techniques are applied: the selected member instances are used in the definition of a convex region of feasibility, following the approach from Chapter 4.

The method performs column generation in accordance with individual member instances. The heuristic is based on the violation of the dual constraint corresponding to $\eta^x_i$ in Equations (4.13,4.14). The variable $\eta^x_i$ appears in several constraints of the primal which themselves involve many other primal variables $\alpha^x_k, \xi^x_i, \xi^x_j$. Therefore, in order to generate the column corresponding to $\eta^x_i$, the heuristic must take their combined effect into account. This involves the solution of a simple linear program derived from the dual constraints. A detailed derivation is given in Appendix A.

The number of member instances that are allowed to influence the solution may be bounded by a parameter $L$. This parameter defines a family of optimization problems with the mixed-integer formulation at one extreme and the disjunctive programming LP approximation at the other extreme. The graded sequence of approximations may be used to characterize how the disjunctive programming approximation effects the consistency of disambiguation and the resulting classifier. By gradually selecting and adding new member instances to the problem, as needed, and removing those from the problem that are unused, the sequence of approximations is expected to be of manageable size.

Therefore, another consequence of the new column generation heuristic is that it is expected to eliminate the need to perform row generation by maintaining LP approximations with few columns. In this case, all parallel reductions (all rows) of the DPBoost formulation may be included in the first approximation. In this case, the sequence of LP approximations are expected to converge more quickly to the solution of the original convex DPBoost problem.

Implementation and empirical results are needed to validate the new approach.

With above modifications, it will be possible to perform a detailed analysis of the behaviour of the disjunctive...
programming approximation applied to LPBoost for learning from ambiguous examples. These experimental design will follow that used in Chapter 4. The over-arching goal is to support my thesis, by showing that specialized algorithms for learning from large sets of ambiguous examples can supplant the need to collect more specific label information.

Furthermore, it is expected that the resulting algorithm(s) will scale to larger problems, and can thus be validated on the existing benchmarks for multi-instance learning. This will further demonstrate the importance of this work.

Finally, the techniques developed will be directly applicable to the work on object recognition.

### 6.3 Learning from Ambiguous Examples for Object Recognition

This work explores an application of learning from ambiguous examples to object recognition. The approach has been validated on simple synthetic data sets. Ongoing work involves scaling the technique to real image data sets.
Bibliography


Appendix A

Practical DPBoost Column Generation

In order to construct a classifier from a set of ambiguous training data with binary labels, the DPBoost formulation from Chapter 4 optimizes a linear objective over a disjunctive constraint set. Then the problem is relaxed using convexification. A new column generation technique is proposed to solve the resulting approximation. A much simpler presentation capturing the essence of the problem is given here. The notations are not necessarily consistent with the remainder of this document.

Assume we have a collection of sets \( X_i = \{x_{i1}, \ldots, x_{iK}\} \) for \( 1 \leq i \leq m \). To avoid subscript clutter, we will refer to members simply as \( x \in X_i \). There is also a distinguished set \( X_0 \) that plays a different role. The optimization problem of interest is

\[
\text{DP} \quad \min_w c^T w \quad \text{(A.1)}
\]

\[
\text{s.t.} \quad \bigvee_{x \in X_i} \left[ (x^T w \geq 1) \land \bigwedge_{x' \in X_0} (x'^T w \geq 1) \land (w \geq 0) \right], \forall i. \quad \text{(A.2)}
\]

We assume that there is a feasible solution. (This is guaranteed by the original learning problem).

A.1 Convexification

The disjunctive sets are individually convexified. Representing the convex hull using only linear constraints requires \textit{lifting}, where we instantiate a separate solution vector \( w^x \in \mathbb{R}^d \) for each pattern \( \forall x \in X_i, \forall i \), and
apply suitable constraints to these \((\text{author?})\) \([6]\). We arrive at the following LP

**DP1**

\[
\begin{align*}
\text{min} & \quad c^T w \\
\text{s.t.} & \quad w^x \geq 0, \quad \eta^x \geq 0, \quad \forall x \in X_i, \forall i \\
& \quad x^T w^x \geq \eta^x, \quad \forall x \in X_i, \forall i \\
& \quad x'^T w^x \geq \eta^x, \quad \forall x' \in X_0, \forall x \in X_i, \forall i \\
& \quad w = \sum_{x \in X_i} w^x, \quad \forall i \\
& \quad 1 = \sum_{x \in X_i} \eta^x, \quad \forall i 
\end{align*}
\]

\((A.3)-(A.8)\)

Notice that the LP constraint matrix has many rows and columns, especially considering that \(d\) may be large.

### A.2 A New Heuristic for Column Generation

In order to use column generation to solve this large LP, we propose a heuristic to select \(d + 1\) columns in parallel - those corresponding to \(\eta^x\) and \(w^x \in \mathbb{R}^d\) - based on the violation of dual constraints. After the adding the columns, the so called restricted master problem is solved and the process is repeated. The intuition for this approach is that, for each set of feature vectors \(X_i\), the expansion \(w = \sum_{x \in X_i} w^x\) for the solution will only involve a few terms, and the remaining variables can thus be left out of the constraint matrix (their convex combination weights \(\eta^x\) will be zero).

#### A.2.1 Dual LP

To select the columns \(\eta^x, w^x\) to add, we consider the interaction of these variables in the dual problem in Equations \((A.9)-(A.13)\). This is written in terms of dual variables

\[
\begin{align*}
u^x & \geq 0, \forall x \in X_i, \forall i \\
\rho_i & \in \mathbb{R}^d, \forall i \\
\theta_i & \in \mathbb{R}, \forall i
\end{align*}
\]

that are introduced for constraints \((A.5)-(A.8)\) respectively.
APPENDIX A. PRACTICAL DPBOOST COLUMN GENERATION

69

\textbf{A.2.2 Constraint Violation}

The primal variables \(\eta^x\) and \(w^x\in\mathbb{R}^d\), are linked to dual constraints A.11 and A.12. Thus, the procedure for selecting new columns is based on the interaction of dual variables \(\rho_i, \theta_i, u^x, u'^x\) appearing in these constraints.

Notice that, after solving the primal, we have access to the duals \(\rho_i, \theta_i\) corresponding to constraints A.7 and A.8. We can also retrieve values for the subset of dual variables \(u^x, u'^x, \forall x \in \mathbb{X}_0\) corresponding to constraints A.5 and A.6 involving \(\eta^x, w^x\) columns that have previously been generated. However, in order to test the violation of the constraint A.11 for a new \(x\), we must infer values \(u^x, u'^x, \forall x' \in \mathbb{X}_0\).

1. For each \(x \in \mathbb{X}_i\), using dual variables \(\theta_i \in \mathbb{R}\) and \(\rho_i \in \mathbb{R}^d\) returned from the primal solver, solve the following LP subproblem, which we call DP1-cg:

\[
\text{DP1-cg} \quad \max_{\theta_i, \rho_i} \sum_{i=1}^m \theta_i \quad \text{s.t.} \begin{align*}
\theta_i &\in \mathbb{R}, \; u^x \geq 0, \; u'^x \geq 0, \; \forall x \in \mathbb{X}_i, \forall i, \forall x' \in \mathbb{X}_0 \\
\theta_i &\leq u^x + \sum_{x' \in \mathbb{X}_0} u'^x, \; \forall x \in \mathbb{X}_i, \forall i \\
u^x x + \sum_{x' \in \mathbb{X}_0} u'^x x' &\leq \rho_i, \; \forall x \in \mathbb{X}_i, \forall i \\
\sum_{i=1}^m \rho_i &\leq c .
\end{align*}
\]

Use the solution to compute the following score \(S(x) = \theta_i - (u^x + \sum_{x' \in \mathbb{X}_0} u'^x)\). If positive, this indicates a violation of the constraint A.11 corresponding to variable \(\eta^x\).

2. Choose \(x^*\) which maximizes \(S(x)\).

(a) If \(S(x^*)\) is positive, add the new columns corresponding to \(\eta^{x^*}, w^{x^*}\) and resolve the primal.

(b) Otherwise, a complete basis and solution to the original has been found.
3. Done.

It is also possible that this strategy could be implemented in conjunction with other (previously implemented) column generation strategies.
Appendix B

List of Symbols Used

- $\mathcal{I}$ - unpreprocessed input space
- $\mathcal{I} \in \mathbb{I}$ - input observation
- $\mathcal{X}$ - preprocessed input space, feature space
- $\mathbb{Y}$ - output space, label space
- $x \in \mathcal{X}$ - instance, feature vector
- $y \in \mathbb{Y}$ - label
- $\mathcal{X} \in 2^\mathcal{X}$, $\mathcal{X} (\mathcal{I}) = \{x_1, \ldots, x_K\}$ - multi-instance, set of feature vectors
- $K = K_\mathcal{I}$ - number of member instances
- $\mathcal{Y} \in 2^\mathbb{Y}$ - multi-instance label set
- $\mathcal{L} : 2^\mathcal{Y} \times \mathbb{Y} \rightarrow \{0, 1\}$ - label semantics, predicate determining labels of multi-instance
- $y_j \in Y \Leftrightarrow \mathcal{L} (\{y_1, \ldots, y_K\} \mathcal{X}, y) = 1$ - how the label semantics work