THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF ELECTRONIC AND INFORMATION ENGINEERING

Machine Learning Approaches for Visual Object Detection

Chensheng SUN

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

August, 2012

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

_____ (Signed)

Chensheng Sun _____(Name of student)

Abstract

Visual object detection is a fundamental and challenging problem in computer vision and image processing. The study of visual object detection usually focuses on two aspects, proposing effective yet efficient features, and designing powerful and fast detectors. While feature extraction is a domain-specific problem for image processing that usually requires substantial knowledge, experience, and even inspiration, designing the detectors usually relies on techniques of pattern recognition and machine learning. In this thesis, we study the machine learning approaches for visual object detection.

We first review several theoretical machine learning issues, in particular, the structural risk minimization learning principle. Then, several empirical loss functions and optimization methods for solving the support vector machines are discussed. Based on insights into the problem formulation and the solving techniques, we propose a generalized forward feature selection scheme that is applicable to a number of problems. For the sake of comprehensiveness, learning problems other than classification, e.g. ranking and structural prediction, are also studied for the purpose of visual object detection.

The Boosting cascade detector is the most popular method in visual object detection. Boosting essentially also falls into the structural risk minimization regime, but with specialized algorithms that treats the problem as a functional optimization. We examine both the functional optimization formulation and the convex optimization formulation of the AdaBoost algorithm, and propose a mixed form that solves the totally-corrective AdaBoost but using weak classifiers derived from the functional optimization perspective. To alleviate the training cost, we propose a feature subset selection method based on the partial least square regression. In building a cascade detector, three important issues are studied, i.e. optimizing for the the asymmetric objective, selecting a proper operating point for each stage of the cascade, and recycling information from the early stages of the cascade. A novel biased-selection strategy for information recycling is proposed.

For the last part of this work, we study the kernel methods. Various methods for improving the efficiency of the kernel scoring function are discussed, and are grouped into three categories, i.e. the approximations of the scoring function, the explicit feature map, and learning a sparse basis set. Inspired by the kernel methods, we propose

a middle-level feature based on the similarity to exemplar instances. For the visual object detection problem, elements from multiple kernel learning and multiple instance learning can be conveniently incorporated into the similarity feature, and a learning framework using the forward feature selection technique and a coarse-to-fine scheme is proposed to learn an efficient visual object detector using the similarity features.

List of Publications

1. Jiwei Hu, Chensheng Sun and Kin Man Lam, "Can a Machine Have Two Systems for Recognition, Like Human Being?", submitted to Pattern Recognition.

2. Chensheng Sun and Kin Man Lam, "A Biased Selection Strategy for Information Recycling in Boosting Cascade Visual Object Detectors", submitted to Pattern Recognition Letters.

3. Chensheng Sun and Kin Man Lam, "Multiple-Kernel, Multiple-Instance Similarity Features for Efficient Visual Object Detection", IEEE Transactions on Image Processing, vol. 22, No. 8, pp. 3050-3061, 2013.

4. Chensheng Sun, Sanyuan Zhao, Jiwei Hu and Kin Man Lam, "Multi-Instance Local Exemplar Comparisons for Pedestrian Detection", 2012 International Conference on Signal Processing, Communications and Computing, pp. 223-227, August 2012.

5. Chensheng Sun, Sanyuan Zhao, Jiwei Hu and Kin Man Lam, "Totally-Corrective Boosting using Continuous-Valued Weak Learners", 2012 IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 2049-2052, March 2012.

6. Chensheng Sun, Jiwei Hu and Kin Man Lam, "Feature Subset Selection for Efficient AdaBoost Training", Proceedings of the 2011 IEEE International Conference on Multimedia and Expo, pp. 1-6, July 2011.

7. Jiwei Hu, Chensheng Sun and Kin Man Lam, "Learning a Discriminative Model for Image Annotation", Proceedings of the 2011 Annual Summit and Conference of the Asia-Pacific Signal and Information Processing Association, October 2011.

8. Jiwei Hu, Chensheng Sun and Kin Man Lam, "Semi-supervised Metric Learning for Image Classification", Proceedings of the Pacific Rim Conference on Multimedia, vol. 6297/2010, pp. 728-735, 2010.

Acknowledgment

I am deeply grateful to my supervisor Prof. Kin-Man Lam. He grants me the opportunity of entering the world of academic research, and provides guidance and insights whenever I need help. I am especially thankful to his support, encouragement, and patience in this prolonged five years of study, and for the very thorough revision he has made of my articles and this thesis.

I would like to thank Prof. Guo Ping Qiu, Prof. Ling Guan, Prof. Xudong Xie, Prof. Cheng Cai, and Prof. Zhanli Sun. The insightful discussion with them has broaden my view and enlightened my mind, and I have learned a lot from their knowledge and experience.

I am fortunate to have the chance to work with Kwok-Wai Wong, Siu-Hong Tse, Wing-Poing Choi, Hei-Sheung Koo, Xiaoguang Li, Guang Feng, Xuejuan Gao, Yue He, Yu Hu, Pengzhang Liu, Sanyuan Zhao, Feng Xu, Guannan Li, Chao Li, Wentao Liu, Deliang Yang, Jiwei Hu, Dong Li, Hailiang Li, Muwei Jian, and Huiling Zhou during my Ph.D. study. I am thankful to their share of knowledge and thoughts at work, and the life we have enjoyed together.

At last, I would express my deep gratitude to my parents, for their endless love and care through my life. No matter how far away, they are always my biggest and best supporters.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 The visual object detection problem

Detecting instances of an object category is one of the most important applications of computer vision. Identifying the existence and location of object instances in an image has direct applications such as optimizing the focus and exposure in digital cameras for human faces, improving automobile driving safety to avoid clashing with other vehicles or pedestrians, etc. Object detection is also the basis of more advanced analysis and image understanding. For example, tracking the trajectory of soccer players for game strategy analysis, analyzing human limb pose for computerized rendering, etc.

(a) (b)

Figure 1.1: Applications of visual object detection. (a) Face detection is now widely used in cameras for improved focus and exposure. (b) Pedestrian detection (using image + radar) is used in vehicles for driving safety.

Though finding objects of a particular category in images is trivial for human vision, the same task is a well-known challenge in computer vision, and has been intensively studied in the recent years. The real world challenges a computerized visual object detection system in several ways. First and foremost, the visual appearance of objects belonging to a semantic category could cover a wide range of variations, i.e. the well-known problem of the semantic gap. Both intrinsic factors and extrinsic factors contribute to the appearance variations. A category usually covers a wide range of different objects. For example, there are numerous types of cars that differ in struc-

ture, surface decoration, etc. Furthermore, many objects are deformable. In particular, the articulated structure is prevalent in nature. An object may look differently due to articulated motion or elastic deformation. In addition, the imaging conditions also has significant impact on how an object looks, for example, the illumination, viewing point, and so on. The second challenge is that most applications require highly accurate detection results, making as few errors as possible. Considering the large amount of data acquired from cameras, and the relatively small number of object instances in the images, the detector should have very low false positive rate while successfully detecting most of the objects. At last, scanning through an image to find an object instance is computational demanding, since large number of object hypothesis need to be tested. Therefore efficiency is crucial for a practical detector, and that is also why visual object detection systems do not come into application until recent years, when fast hardware becomes available at low cost.

Study on visual object detection significantly benefits from the pattern recognition and machine learning research, in which one of the central problems is to estimate a function that produces the desired output for a given input. For visual object detection, the output of the function can be a categorical decision (e.g. true/false for detecting a single class of objects), a confidence score for an object hypothesis, or the expected location of an object in an image, depending on how the object detection problem is formulated. The visual object detection problem also provides a good test bed for old and new machine learning techniques, and promotes the machine learning study by proposing new challenges and abundance of data. We notice that many computer vision researchers are also active in the machine learning field, showing the importance of machine learning in vision tasks.

1.2 State-of-the-art

In this section, we give a review of the state-of-the-art in visual object detection. Many works discuss the problem of finding a specific object in images, subject to various geometric transformation or distortion, and the problem is referred to as near-duplicate object retrieval, and is essentially an image matching problem. The focus of research is usually on interest points, expressive descriptors, e.g. [1, 2, 3, 4], and efficient and robust matching techniques. By contrast, detecting a category of objects needs to deal with much larger within-class variation, and therefore is more challenging.

The object detection problem can be formulated as hypothesis testing. We consider

locating a particular category of objects in an image. A set of object hypotheses $\mathcal{Y}$ can be generated from the image. For example, a hypothesis $\mathbf{y}$ states the existence of that category at a particular location. Image features $\mathbf{x} \in \mathcal{X}$ can be extracted for each hypothesis. Therefore the detector can be represented by a function $f(\mathbf{x}, \mathbf{y})$, that assigns a confidence score or decision to the hypothesis $\mathbf{y}$ according to the image feature $\mathbf{x}$. Usually a non-maximum suppression post-processing step is applied to those accepted object hypotheses, implementing the Occam's razor such that a few accepted hypotheses are sufficient to explain the observed information, which also reflects the heuristics that two objects should not overlap significantly with each other.

In the following, according to how the hypothesis space $\mathcal{Y}$ is formulated, we divide the techniques into two groups: the sliding window technique that simplifies the hypothesis space to rectangular regions is reviewed in section 1.2.1; techniques that utilize assumptions specific to the visual objects are reviewed in section 1.2.2. Finally in section 1.2.3, other issues studied in visual object detection are introduced.

1.2.1 The sliding window paradigm

In its simplest form, the hypothesis set $\mathcal{Y}$ consists of rectangular windows in the image. This simplistic construction brings two benefits. First, it is straightforward to explore the hypothesis space by nearly exhaustive evaluation of the hypothesis set, and the detector works as a window that slides over the image. Second, it is convenient to design a feature space for the rectangular windows. Therefore the sliding window paradigm is the most widely used method in literature. The study on sliding window detection focuses on two aspects, i.e. image features and classifiers. While the later is a general machine learning problem, the former extensively uses domain knowledge specific to computer vision.

Image features

A rectangular window in an image can be represented by the array of pixel values, and some works directly utilize the pixel values, e.g. [5, 6, 7]. For detecting objects in video, a motion vector can be computed for each pixel to obtain a motion map, e.g. [8]. More complex features can be extracted based on the array of per-pixel values to improve the discriminative power and invariance to intra-class variation. Intuitively, the following information can be helpful for visual object detection in addition to the per-pixel values: (1) an unordered distribution of the per-pixel values, e.g. a histogram;

(2) local contour shape or layout of the pixels; (3) repetitive texture patterns, e.g. the Fourier spectrum. To capture these information, extensive works have been devoted to designing image features. The feature extraction process usually consists of a number of elemental operations, which we summarize as follows.

First, basic operators such as linear filtering, thresholding, quantization, max, min, etc. can be applied to the per-pixel values. The operator may have various response fields, i.e. the set of pixels involved in the operator. For example, gradient can be computed for each pixel using the adjacent pixels, the local binary pattern [9] binary code is obtained by comparing two pixels and then quantizing to $\{0, 1\}$, while the Gabor feature [10, 11] and the Haar-like features [12, 13] are calculated using all pixels in a local region. The granule comparison features [14, 15] consider arbitrary pairs of regions, and the self-similarity feature [16, 17] measures the rough layout of objects by the similarity/difference between pairs of regions. Max pooling is used in [10, 11] to mimic the visual cortex in biological vision systems.

Second, an encoding scheme can be employed to introduce semantic meanings. The encoding scheme can be manually specified, such as the LBP [9] encoding scheme that introduces the notion of uniformity to distinguish structured regions and cluttered regions. More commonly a codebook is used, e.g. obtained by k-means clustering. The encoding can be a simple assignment to the codewords, or encoded as a vector whose length is equal to the size of the codebook, e.g. soft assignment to the codewords, sparse coding, and the locality constrained encoding [18].

At last, statistics can be computed for a set of image elements to give an orderless summarization and invariance of small deformation. A histogram can be used to represent the occurrence frequency, e.g. the color histogram, gradient orientation histogram [19], the local binary pattern histogram [9], the bag-of-visual-words histogram [20], etc. Simple histograms can be extended by adding more dimensions, considering the interaction of the image elements. For example, gray-level co-occurrence is a classic texture descriptor [21], and is extended for color images in [22]. A recent work [23] extends the co-occurrence histogram to include more information, e.g. color difference of pixels. The co-occurrence feature is generalized for visual codewords as in [24], such that the feature is semantically more meaningful. Second order statistics such as the covariance matrix has been successfully used in [25, 26].

Besides, we note that the image can also be represented as a disjoint set of elements rather than a feature vector. For example, the contour fragment is used in [27, 28, 29, 30, 31]. Directly learning from these feature representation requires operators that work on sets, for example, the Chamfer matching for measuring the similarity of

contours.

The context of a window also constitutes an important cue for the existence of an object, especially when the cue from the window itself is weak. A lot of work has been devoted to the context feature in recent years, e.g. [32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. Various types of context can be employed, for example, the global scene category, the location of the object in the scene, the interaction of the object and other objects in the scene, etc. To apply the context cue, we may directly include it in testing the hypothesis, prune the hypothesis space before detection, or enhance the score of a hypothesis in post-processing of the classifier outputs.

Classifiers

Given a feature representation, learning the classifier is a problem extensively studied in the machine learning community. Probabilistic approaches have been adopted in some works, e.g. [27] combines the responses of multiple templates using a Bayesian approach. Many other successful works in visual object detection are based on state-of-the-art machine learning techniques, among which the most popular are the support vector machines (SVM) and the Boosting methods.

Before we introduce the learning algorithms, we note that a challenge in learning a good classifier for object detection is the very high dimensionality of the feature vector. For example, hundreds of thousands of Haar-like features can be enumerated for a window, and the associated pair of granule comparison feature [15] easily produces a feature space of millions and even billions of dimensions due to the combinatorial explosion. Therefore feature selection and dimensionality reduction is crucial. Good performance can be achieved using simple learning algorithm with effective dimensionality reduction. For example the partial least square [42, 43] approach effectively finds a low-dimensional subspace and then good performance is achieved using simple quadratic discriminant analysis. Other algorithms perform feature selection such that the feature extraction cost can also be reduced, e.g. Boosting [12] and random forests [44].

Built upon the statistical learning theory [45], the SVM guarantees good generalization performance on new data by balancing between minimizing the VC dimension of the classifier and minimizing the empirical loss. The method is further enhanced by the kernel trick which introduces an implicit feature space, and allows object representation other than in a vector space, e.g. representing an object as a disjoint set of local features, as long as a kernel can be defined on pairs of object instances. A lot of im-

provements have been made for the SVM, and is still an active research topic. Various problems are solved by the SVM, e.g. regression, multi-class classification [46], ranking [47], structural prediction [48], etc., and many of them quickly find application in visual object detection. New optimization methods have been proposed for training SVM such that large scale dataset can be learned efficiently, e.g. [49, 50, 51]. The performance of SVM is enhanced by kernels that are more suitable to the problem. Kernels tailored for the vision problems have been proposed, e.g. the pyramid matching kernel [52], the spatial pyramid matching kernel [53], and the efficient matching kernel [54]. Furthermore, good kernels can be learned instead of manually designed, e.g. by multiple kernel learning (MKL) [55]. The efficiency of the classifier can be improved by various methods, such as feature selection [56, 57, 58], reducing the number of support vectors [59, 60, 61, 62], and using approximate feature maps [63, 64].

Boosting [65] was developed in an attempt to combine a number of weak learners to obtain a strong learner. It is of significant interest in visual object detection for two reasons. First, a powerful classifier can be learned without overfitting, which may stem from the high dimensional feature space. Second, the resulting classifier can be very efficient to evaluate, meeting the demand of real-time applications. The Boosting algorithm itself has been analyzed by many works, to understand its properties and to improve its performance. For example, Boosting is interpreted as as additive regression in [65], from the margin maximization learning perspective in [66], and as a convex optimization problem in [67, 68]. Essentially the Boosting algorithm can be unified into the structural risk minimization learning paradigm as SVM, but is solved by a step-wise approach that incrementally generates features in the form of the selected weak learners. The Boosting algorithm is tailored for the visual object detection problem in many works. For example, cost-sensitive loss is introduced in [69, 70, 71] such that the classifier is optimal for unbalanced performance requirement. Chained cascade of boosting classifiers are studied in [72, 73, 74, 75], such that the information used in the previous stages are also exploited in the later boosting classifiers, and the resulting detector is more efficient to evaluate.

1.2.2 Methods considering properties of visual objects

Though the sliding window approach is simple and powerful, it ignores crucial information about the object. For example, many objects are constituted by meaningful parts in a meaningful configuration. Methods that directly consider the properties of visual object categories are also well studied in literature.

A visual object can be divided into parts, and naturally each part contributes to confidence score of the full object. An approach directly using this idea is the generalized Hough transform [7, 76, 77, 44], where a detected part casts votes for a number of object hypotheses according to the displacement between the part location and the object location. The parts are limited to interest points and a codebook of parts is learned via clustering in [7, 76, 77], while in [44] the parts are densely sampled and the codebook and votes are learned by a random forest. The Hough transform is essentially a linear model as described in [77]. It is also interpreted as a star shape probabilistic graphic model called the Markov random field. More complex graphs can be used, e.g. the constellation model [78, 79]. The parameters of the Markov random field can be learned using maximum likelihood, or using structural SVM [48]. The part models can be learned using discriminative methods, as in the discriminatively trained part models [80]. The idea of part based model is further enhanced to obtain a grammar model in [81], which allows more flexible combination of the parts.

For the discriminatively trained part models [80], the location of each part needs to be decided in order to maximize the object score, i.e. the object hypothesis is extended as $\mathbf{y} = [\mathbf{y}_0, \mathbf{y}_1, ...\mathbf{y}_n]$, where $\mathbf{y}_0$ is the status (e.g. location, size, etc.) of the full body, and $\mathbf{y}_i$ is the status of the i-th part. Therefore efficient techniques are necessary in order to explore this high dimensional hypothesis space, for example, the belief propagation method.

Due to the use of a detailed hypothesis space and the use of part information, these methods conveys extra information about the object, which can be readily extracted from the detection result, and utilized in higher level analysis, e.g. segmenting the object from background [76], estimating the status of the object [82], and so on.

1.2.3 Other issues in visual object detection

Many other learning issues are encountered in the context of visual object detection. In this section we briefly introduce three of them, i.e. improving the efficiency of the detector, learning from weakly labeled data, and scaling up the detector to multiple classes.

Detector efficiency

For a typical image, the hypothesis set $\mathcal{Y}$ usually consists of a large number of hypotheses. Even with the sliding window approach working on a regular grid of windows, the number of windows easily reaches hundreds of thousands in a typical image. To im-

prove the efficiency of the detector, various methods have been devised. First, efficient methods for feature extraction have been studied, e.g. using the integral image [12], caching the features and sharing them between windows, etc. Second, attention-based mechanism can be employed to quickly reduce the search space. For example, early works based on interest point detectors [79] limit the hypothesis set to those supported by sufficient number of interest points, and generalized Hough transform [77] quickly get the scores for all hypotheses in a feature-centric manner—instead of collecting the confidence for each hypotheses, each observed image feature casts votes into the hypothesis space. The sparsity of the votes significantly reduces the computational cost. The cascade approach [12] is another attention-based approach. The cascade is composed of a number of classifiers, usually with increasing complexity as moving downwards the cascade. Most non-object hypothesis can be rejected by the first few stages, incurring a very small amount of computational cost, and only those hypotheses likely to be objects need to be checked by the later stages, which are more powerful but also more complex. Numerous improvements have been incorporated into the cascade approach, e.g. [83]. Recently the attention-based cascade has been generalized for simultaneously detecting multiple classes of objects, by relying on the general idea of objectness, e.g. [84, 85, 86]. The recent branch-and-bound approach [87] is another solution to avoid checking all hypotheses. It branches the hypothesis set into subsets, and calculates a bound on the maximum score for each subset. Those subsets with insufficient upper bounds can be rejected without checking any hypothesis in them.

Weakly labeled data

Learning the object detector from training data is usually formulated as a supervised learning problem. Each training instance is given in the form of a pair of image and label $(I_i, \mathbf{y}_i)$, and the feature $\mathbf{x}_i$ is extracted from the image according to the label $\mathbf{y}_i$. However, the training instances could be weakly labeled. First, the label may not be accurate. For example, it is hardly possible to define the bounding box exactly and consistently by different human labelers, and inevitably there is some alignment error between different training instances. Second, the labels may not be fully provided for an training instance. For example, we want to learn a part-based detector, and the label $\mathbf{y} = [\mathbf{y}_0, \mathbf{y}_1, ...\mathbf{y}_n]$ should consist of the full body label $\mathbf{y}_0$ and the part labels $\{\mathbf{y}_i; i > 0\}$, but it is often the case that only the full body label is provided. Third, the labels could be missing for some training instances, and we are given both labeled and unlabeled data. Various techniques are proposed to deal with these cases. The inac-

curate labels can be treated in the multiple instance learning problem [88, 89, 90, 91], which generates a bag of instances for each original instance, and assumes that the bag includes the optimal label. Learning a part-based detector only using holistic labels is studied in [80], where the parts are initialized using heuristics and then iteratively refined. Semi-supervised learning [92] solves the case of missing labels for some training instances, i.e. learning from both labeled and unlabeled data. The unlabeled data participates in the learning process in various manners, e.g. as additional regularization terms. Adapting a pre-trained detector to a particular environment also deals with missing labels of the new obtained data, e.g. adapting a generic pedestrian detector to optimize performance for surveillance video in a particular crossroad [93].

Scale up to many classes

Though many application systems only require detecting a single category of objects, e.g. face detectors in cameras and pedestrian detectors for surveillance video, it is still desirable to be able to detect many classes of objects. Detecting many classes brings new difficulties in several aspects. A naive implementation would cause the complexity of the system grows linearly with the number of classes. To alleviate this problem, methods have been proposed such that when training the detectors for several classes together, the detectors may share computation among the classes [94] and thus the complexity of the full system increases sub-linearly. Class-independent measures such as "objectness" have been studied in several works [84, 86] such that a window can be decided to be an object or background without knowing which object category it is. For the generalized Hough transform and discriminative part-based models, the knowledge sharing can be easily implemented by sharing the codebook or sharing parts between classes, e.g. [95, 96]. Besides the detector efficiency issue, sharing/transferring knowledge between different classes has also been studied for two other reasons. First, the accuracy for classes with small number of available training instances can be improved by borrowing strength from other classes, which is studied as a multi-task learning problem; second, the human labeling effort for learning new classes can be significantly reduced, since a few examples are sufficient for learning a good classifier. Direct model transfer is studied in [97, 98, 99], while the attribute [100, 101] is introduced as an intermediate concept for knowledge sharing.

1.3 Scope of this thesis

In this thesis, we study a basic problem in visual object detection, i.e. how to train a good visual object detector. We adhere to the sliding window paradigm in most parts, and focus on the machine learning techniques. The goal is to improve performance, efficiency, and training speed. The main body of this thesis can be divided into the following three parts.

The structural risk minimization (SRM) learning principle is studied in Chapter 2. Many techniques use the spirit of structural risk minimization, for example, support vector machines, logistic regression, and Boosting. A number of learning problems can be solved, e.g. classification, ranking, structural prediction, etc. The properties and solving techniques of several problems are analyzed. Based on the in-depth understanding of the SRM problems, a forward feature selection method is proposed such that a small feature subset can be found while good performance can be maintained, and efficient classifiers can be built using this technique.

Chapter 3 is devoted to the Boosting algorithm, which is one of the most successful techniques in contemporary applications. We study the theory and technical details of for training Boosting classifiers, and improve the existing methods in two points. First, the totally-corrective Boosting methods are studied and a second-order criterion for weak classifier selection is incorporated into the totally-corrective Boosting. Second, a feature subset selection method based on the partial least square regression is proposed to accelerate the training of the Boosting classifiers.

The Boosting algorithm is popular in visual object detection due to its successful marriage with the cascade detector. Therefore, Chapter 4 is devoted to the Boosting cascade detector. We study several important issues in training the Boosting cascade detector. First, each stage of a cascade is a classifier with a highly unbalanced performance, and how to adapt normal learning algorithms for this asymmetric performance goal should be brought to attention. Second, setting the performance goal for each stage of the cascade is a key to detector efficiency, but is usually neglected in research works. Lastly, we propose methods for further improving the cascade detector's efficiency by re-using information that is already obtained in the earlier stages of the cascade.

In Chapter 5, kernel methods are studied, and various methods for improving the efficiency of the kernel methods are discussed. Inspired by the kernel methods, we propose a middle-level feature based on the similarity to exemplar instances. For the visual object detection problem, we incorporate elements from multiple kernel learn-

ing and multiple instance learning into the similarity feature, and propose a learning framework using the forward feature selection technique and a coarse-to-fine scheme for finding good exemplars.

Chapter 2 Structural risk minimization learning

In this chapter we study the problem of learning a classifier from labeled training data. We review the structural risk minimization principle in Section 2.1, focusing on why and how we can avoid overfitting and obtain a classifier with low error rate on testing data. Then several loss functions for binary classification are studied in Section 2.2, entailing three popular techniques, i.e. support vector machine (SVM), logistic regression, and AdaBoost. Techniques for solving the learning problems are discussed in Section 2.3, taking the SVM as an example. Based on analysis of the solving techniques, a generalized forward feature selection algorithm is proposed in Section 2.4. Finally in Section 2.5 we discuss the ranking and structural prediction problems, which also find successful applications in visual object detection.

2.1 Structural risk minimization

2.1.1 Learning to minimize the expected loss

Let each sample instance be a labeled pair drawn from an unknown distribution, i.e. $(\mathbf{x}, y) \sim D$, where $\mathbf{x}$ is the feature vector, y is the label, and D is the joint distribution over features and labels. Let $\mathcal{X}$ be the domain of x, and $\mathcal{Y}$ be the domain of y. In the following we consider binary classification problems in a N-D vector space, i.e. $\mathcal{X} = \mathbb{R}^N$, $\mathcal{Y} = \{+1, -1\}$. Given a training set S with $M = |S|$ labeled instances $(\mathbf{x}, y)$, we want to learn a function f that predicts the label $\hat{y}$ for an instance with feature $\mathbf{x}$, i.e. $\hat{y} = f(\mathbf{x})$. The function f is selected from a set $\mathcal{H}$ of functions, which can contain a finite or infinite number of functions. The goal is to select a function that has low expected error rate on the distribution D, i.e. low expected loss, also called the generalization error. The problem of generalization has been studied from various aspects in the statistical learning theory, e.g. Bayesian learning, stability theory, etc. In the following we briefly introduce the rationale of learning using the concept of VC dimension [45]. The VC (Vapnik-Chervonenkis) dimension of a function set $\mathcal{H}$ is

defined as follows:

Definition. (VC dimension) *A set S of instances can be **shattered** by $\mathcal{H}$, if for **arbitrary** labeling of these instances, there **exists** a function in $\mathcal{H}$ that correctly predicts the labels for **all** instances in S. The VC dimension d of $\mathcal{H}$ is the size of the **largest** set S that can be shattered by $\mathcal{H}$, i.e. there exists a set of size d that can be shattered by $\mathcal{H}$, but there does not exist a set of size $d+1$ that can be shattered by $\mathcal{H}$.*

Based on the VC dimension, a probabilistic upper bound on the expected loss (generalization error) is given in [45] as the following theorem:

Theorem. *Denote the expected loss (generalization error) as $err_D = P_{(\mathbf{x},y)\sim D}\left(f\left(\mathbf{x}\right) \neq y\right)$, and denote the empirical loss (training error) as $err_S = P_{(\mathbf{x},y)\sim S}\left(f\left(\mathbf{x}\right) \neq y\right)$. Then with probability $1-\eta$, the expected loss is upper bounded by the following expression:*

$$err_D \leq err_S + \sqrt{\frac{d\left(\log\left(2M/d\right)+1\right) - \log\left(\eta/4\right)}{M}}$$

Though tighter bounds can be found for various classifiers, this bound is already quite instructive. It suggests that in order to achieve good generalization performance, we should strike a balance between the empirical loss and the complexity of the function set $\mathcal{H}$, indicated by the VC dimension d. Usually the following problem is solved for learning from training data:

$$\min_{f} \quad L\left(f\right) \quad \text{s.t.} \quad R\left(\mathcal{H}\right) = c. \tag{2.1}$$

It can also be written in the unconstrained Lagrangian form:

$$\min_{f} \quad L\left(f\right) + \lambda R\left(\mathcal{H}\right) \tag{2.2}$$

with one-to-one correspondence between c and λ. $R\left(\mathcal{H}\right)$ is the model complexity term, e.g. the VC dimension of $\mathcal{H}$, and it constrains the search for a good classifier in a function set with limited complexity. $L\left(f\right)$ is the empirical loss term, requiring the classifier to have low error on the training data. In formulation (2.1), the model complexity is fixed by constraining to a particular $\mathcal{H}$, and the empirical loss is minimized by finding the best function in $\mathcal{H}$. Then model selection is required to find the optimal model complexity to minimize the expected loss, by trying different $\mathcal{H}$, which is equivalent to varying c or λ, and the best c or λ can be determined by minimizing the expected loss on a standalone validation set.

2.1.2 Large margin linear classifiers

The linear classifiers are the most popular classifiers in practice. Without loss of generality, we consider the separating hyperplane passing through the origin, and with unit normal vector $\mathbf{w}$:

$$f(\mathbf{x}) = sign\left(\mathbf{w}^T\mathbf{x}\right), \|\mathbf{w}\|_2 = 1$$

The VC dimension of the set of linear classifiers in $\mathbb{R}^N$ is $N+1$. The linear classifier is popularized by the max-margin learning, which shows that the VC dimension of linear classifiers can be effectively bounded. Define the margin of an example $(\mathbf{x}, y)$ to be $y\mathbf{w}^T\mathbf{x}$, and define the Δ-margin separating hyperplanes to be a linear classifier that separates the positive and negative data by margin Δ, i.e. $y\mathbf{w}^T\mathbf{x} \geq \Delta$. The following theorem from [45] gives an generalization error bound for large margin classifiers:

Theorem. *Let vectors $\mathbf{x} \in \mathcal{X}$ be within a sphere of radius R, the set of Δ-margin separating hyperplanes has VC dimension d bounded by the following inequality:*

$$d \leq \min\left(\left[\frac{R^2}{\Delta^2}\right], N\right) + 1.$$

Then with probability $1-\eta$, the expected loss is upper bounded by:

$$err_D \leq \frac{m}{M} + \frac{\varepsilon}{2}\left(1 + \sqrt{1 + \frac{4m}{M\varepsilon}}\right),$$

where $\varepsilon = 4\frac{d(\log(2M/d)+1)-\log(\eta/4)}{M}$, and m is the number of misclassified training instances.

Therefore the learning problem can be transformed to minimizing the empirical loss for a fixed model complexity measured by the margin:

$$\min_{\mathbf{w}} \quad L(f) \quad \text{s.t.} \quad \Delta = c, \ \|\mathbf{w}\|_2^2 = 1, \tag{2.3}$$

which is equivalent to:

$$\min_{\mathbf{w}} \quad L(f) \quad \text{s.t.} \quad \Delta = 1, \ \|\mathbf{w}\|_2^2 = \sqrt{1/c}\,, \tag{2.4}$$

or in the unconstrained form with one-to-one correspondence between λ and c:

$$\min_{\mathbf{w}} \quad \lambda\|\mathbf{w}\|_2^2 + L(f) \quad \text{s.t.} \quad \Delta = 1. \tag{2.5}$$

2.1.3 L-1 norm regularization

The max-margin separating hyperplane is formulated using unit normal vector $\mathbf{w}$, which results in L-2 norm regularization in problem (2.5). L-1 norm regularization is also a popular technique. A generalization error bound for L-1 norm regularization is given in the Boosting literature [66]:

Theorem. *Let $\mathcal{H}$ be the set of base classifiers (or weak classifiers, weak hypotheses), and with VC dimension d. Let $f(\mathbf{x}) = \sum_{i=1}^{T} \alpha_i h_i(\mathbf{x})$, $\sum_{i=1}^{T} |\alpha_i| = 1$ be the strong classifier obtained by a linear combination of T weak classifiers from $\mathcal{H}$. Then with probability $1 - \eta$, the following upper bound on the expected loss holds:*

$$err_D \leq P_S(yf(\mathbf{x}) \leq \Delta) + O\left(\frac{1}{\sqrt{M}}\sqrt{\frac{d\log^2(M/d)}{\Delta^2} + \log\frac{1}{\eta}}\right) \tag{2.6}$$

The first term on the right hand side of (2.6) approximately measures error on the training set, and the second term is determined by the margin Δ. The bound implies that we should minimize the empirical loss and maximize the margin, but with L-1 norm constraint for $\mathbf{w}$. Furthermore, the formulation is independent of $\mathbf{T}$. Therefore we may allow all the weak hypotheses to be included in the problem, but indeed only a subset of base classifiers will be selected due to the sparsity-inducing L-1 norm constraint on the linear combination coefficient $\boldsymbol{\alpha}$.

2.2 Loss functions for classification

The bound on the expected loss is determined by both the model complexity and the empirical loss. As shown in section 2.1, the model complexity of linear classifiers can be bounded using the concept of margin, by regularizing the L-1 or L-2 norm of the normal vector $\mathbf{w}$ of the separating hyperplane. Therefore another problem is how to calculate the empirical loss such that we have a problem that can be conveniently solved.

Usually the empirical loss is represented as sum of loss on the training instances:

$$L(f) = \sum_{i=1}^{M} l(y_i, f(\mathbf{x}_i)),$$

where $l(y_i, f(\mathbf{x}_i))$ is the loss per training instance, and ideally the 0-1 loss should be

used for binary classification problems:

$$l\left(y, f\left(\mathbf{x}\right)\right) = \begin{cases} 0 & y = sign\left(f\left(\mathbf{x}\right)\right) \\ 1 & y \neq sign\left(f\left(\mathbf{x}\right)\right) \end{cases}$$

However, the 0-1 loss is not convex and not differentiable, making the learning problem have many local optima and difficult to solve. In the following we discuss three approximations of the 0-1 loss, i.e. the hinge loss used in the support vector machines, the logistic loss used in logistic regression, and the exponential loss used in AdaBoost. The loss functions are shown in Fig. (2.1). We can see all are upper bound of the 0-1 loss.

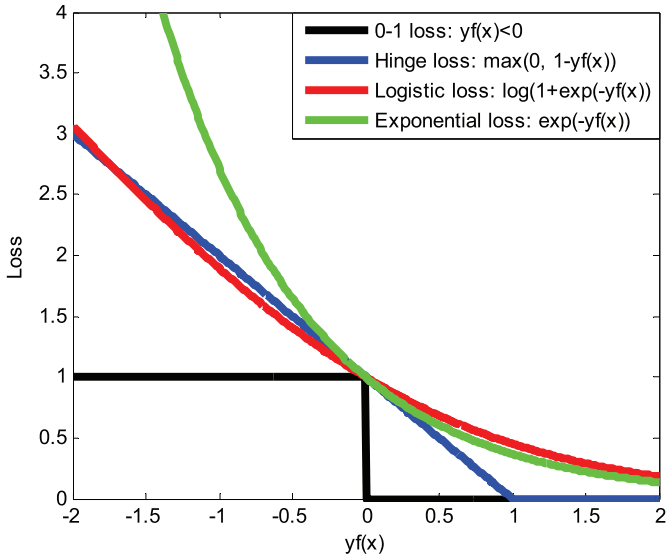


Figure 2.1: Loss functions for binary classification.

2.2.1 Hinge loss

The hinge loss is formulated as follows:

$$l\left(y, f\left(\mathbf{x}\right)\right) = \max\left(0, 1 - yf\left(\mathbf{x}\right)\right)^{\sigma}, \text{with small } \sigma > 0.$$

As σ approaches 0, the hinge loss approximates the 0-1 loss with increasing accuracy. However, usually $\sigma = 1$ or $\sigma = 2$ are used since they lead to problems easy to solve. The hinge loss is used in the support vector machines. For L-2 regularized SVM and

$\sigma = 1$, the learning problem is:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M} \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right) \tag{2.7}$$

Introducing the slack variables $\{\xi_i\}$, the problem can be transformed to:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i,\ i = 1, ..., M, \\ & \xi_i \geq 0,\ i = 1, ..., M. \end{aligned} \tag{2.8}$$

The hinge loss introduces interesting properties that gives the method its name "support vector machine". We study its properties by the Lagrangian dual problem. The Lagrangian of (2.8) is:

$$\mathcal{L}\left(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i + \sum_{i=1}^{M}\alpha_i\left(1 - \xi_i - y_i\mathbf{w}^T\mathbf{x}_i\right) - \sum_{i=1}^{M}\beta_i\xi_i \tag{2.9}$$

The following Karush-Kuhn-Tucker (KKT) conditions are sufficient and necessary for the optimality of the original problem:

Stationary	Primal and dual feasibility	Complementary slackness
$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M}\alpha_i y_i\mathbf{x}_i = 0$	$y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$	$\alpha_i\left(1 - \xi_i - y_i\mathbf{w}^T\mathbf{x}_i\right) = 0$
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} = C - \boldsymbol{\alpha} - \boldsymbol{\beta} = 0$	$\xi_i \geq 0$	$\beta_i\xi_i = 0$
	$\alpha_i \geq 0$	
	$\beta_i \geq 0$	

Substituting the KKT conditions into the Lagrangian, the dual problem can be obtained, which is a quadratic programming problem with box constraints:

$$\min_{\boldsymbol{\alpha}} \quad \tfrac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C,\ i = 1, ..., M, \tag{2.10}$$

where Q is a $M \times M$ matrix, $Q_{ij} = y_i y_j \mathbf{x}_i^T\mathbf{x}_j$, and $\mathbf{e}$ is a vector of all 1s. The following relations can be obtained from the KKT conditions:

- $\mathbf{w} = \sum_{i=1}^{M}\alpha_i y_i\mathbf{x}_i$.
- For those out-margin examples: $y_i\mathbf{w}^T\mathbf{x}_i > 1$, $\xi_i = 0$, $\alpha_i = 0$, $\beta_i = C$.
- For those on-margin examples: $y_i\mathbf{w}^T\mathbf{x}_i = 1$, $\xi_i = 0$, $0 \leq \alpha_i \leq C$, $\beta_i = C - \alpha_i$.

- For those in-margin examples: $y_i \mathbf{w}^T \mathbf{x}_i < 1$, $\xi_i > 0$, $\beta_i = 0$, $\alpha_i = C$.

Therefore the separating hyperplane is decided only by those on-margin and in-margin examples, i.e. $y_i \mathbf{w}^T \mathbf{x}_i \leq 1$. The on-margin and in-margin examples are called the support vectors. However, there is no closed form relation to determine the $\{\alpha_i\}$ for those on-margin examples (i.e. those $y_i \mathbf{w}^T \mathbf{x}_i = 1$), since the hinge loss function is not differentiable at $yf(\mathbf{x}) = 1$.

2.2.2 Logistic loss

Assume the likelihood of a labeled instance $(\mathbf{x}, y)$ is $(1 + \exp(-yf(\mathbf{x})))^{-1}$. If the margin $yf(\mathbf{x})$ is a large positive value, the probability approaches 1, if the margin is a large negative value, the probability approaches 0. Then the negative log likelihood can be used as a loss function:

$$l(y, f(\mathbf{x})) = \log(1 + \exp(-yf(\mathbf{x}))). \tag{2.11}$$

The logistic loss $L(f) = \sum_{i=1}^{M} \log(1 + \exp(-y_i f(\mathbf{x}_i)))$ is the negative log likelihood of the training set, and minimizing the logistic loss is equivalent to the maximum likelihood estimation of the parameter of the distribution.

The population minimizer of the logistic loss can be derived in closed form. We minimize the expected loss functional with respect to the distribution D:

$$\min_f \quad E_D(\log(1 + \exp(-yf(\mathbf{x})))) \tag{2.12}$$

The gradient of the loss functional is:

$$\begin{aligned} E_D\left(-\tfrac{y\exp(-yf(\mathbf{x}))}{1+\exp(-yf(\mathbf{x}))}\right) = \quad & -P_D(y=1,\mathbf{x})\tfrac{\exp(-f(\mathbf{x}))}{1+\exp(-f(\mathbf{x}))} \\ & +P_D(y=-1,\mathbf{x})\tfrac{\exp(f(\mathbf{x}))}{1+\exp(f(\mathbf{x}))} \end{aligned} \tag{2.13}$$

By setting the gradient to 0, we can obtain the solution of problem (2.12) as:

$$f(\mathbf{x}) = \log\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} \tag{2.14}$$

(2.14) is called the logit of the probability $P(y=1|\mathbf{x})$. Minimizing the logistic loss using a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is known as the logistic regression. The L-2 norm

regularized logistic regression problem is:

$$\min_{\mathbf{w}} \quad \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \log\left(1 + \exp\left(-y_i\mathbf{w}^T\mathbf{x}_i\right)\right) \tag{2.15}$$

Introducing the auxiliary variables $m_i = y_i\mathbf{w}^T\mathbf{x}_i$, the problem becomes:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{m}} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \log\left(1 + \exp\left(-m_i\right)\right) \\ \text{s.t.} \quad & m_i = y_i\mathbf{w}^T\mathbf{x}_i,\, i = 1, ..., M. \end{aligned} \tag{2.16}$$

The Lagrangian of (2.16) is:

$$\mathcal{L}\left(\mathbf{w}, \mathbf{m}, \boldsymbol{\alpha}\right) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \log\left(1 + \exp\left(-m_i\right)\right) + \sum_{i=1}^{M} \alpha_i\left(m_i - y_i\mathbf{w}^T\mathbf{x}_i\right), \tag{2.17}$$

and the KKT conditions are:

Stationary	Primal and dual feasibility	Complementary slackness
$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \lambda\mathbf{w} - \sum_{i=1}^{M}\alpha_i y_i\mathbf{x}_i = 0$	$m_i = y_i\mathbf{w}^T\mathbf{x}_i$	
$\frac{\partial\mathcal{L}}{\partial m_i} = \frac{-\exp(-m_i)}{1+\exp(-m_i)} + \alpha_i = 0$		

The dual problem is an unconstrained problem:

$$\min_{\boldsymbol{\alpha}} \quad \tfrac{\lambda}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i=1}^{M}\left(1 - \alpha_i\right)\log\left(1 - \alpha_i\right) + \sum_{i=1}^{M}\alpha_i \log\alpha_i \tag{2.18}$$

Similar to the SVM, the separating hyperplane is in the subspace spanned by the training instances:

$$\mathbf{w} = \frac{1}{\lambda}\sum\nolimits_{i=1}^{M}\alpha_i y_i\mathbf{x}_i, \tag{2.19}$$

using the dual variables $\{\alpha_i\}$ as the combination weights. But the difference is, since the logistic loss is differentiable, a closed form relation between the primal and dual variables exists for the logistic loss:

$$\alpha_i = \frac{\exp\left(-m_i\right)}{1 + \exp\left(-m_i\right)}, \tag{2.20}$$

In particular, the dual variables $\{\alpha_i\}$ are equal to the negative gradient of the primal

objective w.r.t. the margins $\{m_i\}$ of the training instances:

$$L = \sum_{i=1}^{M} \log\left(1 + \exp\left(-m_i\right)\right), \text{and } \alpha_i = -\frac{\partial L}{\partial m_i}. \tag{2.21}$$

2.2.3 Exponential loss

The exponential loss, popular due to its use in the AdaBoost algorithm, has the following form:

$$l\left(y, f\left(\mathbf{x}\right)\right) = \exp\left(-yf\left(\mathbf{x}\right)\right) \tag{2.22}$$

Similar to the logistic loss, the exponential loss is convex and differentiable, and the population minimizer of the exponential loss can also be derived in closed form. Given the distribution D, we minimize the expected loss functional on D:

$$\min_{f} \quad E_D\left(\exp\left(-yf\left(\mathbf{x}\right)\right)\right) \tag{2.23}$$

The gradient of the expected loss is:

$$\begin{aligned} E_D\left(-y\exp\left(-yf\left(\mathbf{x}\right)\right)\right) = \quad & -P_D\left(y=1,\mathbf{x}\right)\exp\left(-f\left(\mathbf{x}\right)\right) \\ & +P_D\left(y=-1,\mathbf{x}\right)\exp\left(f\left(\mathbf{x}\right)\right) \end{aligned} \tag{2.24}$$

Set the gradient to 0, and we can obtain the solution as:

$$f\left(\mathbf{x}\right) = \frac{1}{2}\log\frac{P\left(y=1|\mathbf{x}\right)}{P\left(y=-1|\mathbf{x}\right)} \tag{2.25}$$

The solution (2.25) is indeed the half logit, while the logistic loss results in a solution of the full logit (2.14).

We also study the property of learning with exponential loss by studying its Lagrangian dual problem. A linear classifier is learned by solving the following problem:

$$\min_{\mathbf{w}} \quad \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M}\exp\left(-y_i\mathbf{w}^T\mathbf{x}_i\right) \tag{2.26}$$

Introducing the auxiliary variables $m_i = y_i\mathbf{w}^T\mathbf{x}_i$, the problem becomes:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{m}} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M}\exp\left(-m_i\right) \\ \text{s.t.} \quad & m_i = y_i\mathbf{w}^T\mathbf{x}_i,\, i = 1,...,M. \end{aligned} \tag{2.27}$$

The Lagrangian is:

$$\mathcal{L}(\mathbf{w}, \mathbf{m}, \boldsymbol{\alpha}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \exp(-m_i) + \sum_{i=1}^{M} \alpha_i \left(m_i - y_i \mathbf{w}^T \mathbf{x}_i\right), \tag{2.28}$$

and the KKT conditions are:

Stationary	Primal and dual feasibility	Complementary slackness
$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \lambda \mathbf{w} - \sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i = 0$	$m_i = y_i \mathbf{w}^T \mathbf{x}_i$	
$\frac{\partial \mathcal{L}}{\partial m_i} = \alpha_i - e^{-m_i} = 0$		

The dual problem is:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2\lambda}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \sum_{i=1}^{M} \alpha_i (1 - \log \alpha_i) \tag{2.29}$$

Similar to the case of logistic loss, there exists a closed form relation between the primal and dual variables, and the dual variables are equal to the negative gradient of the loss function:

$$L = \sum_{i=1}^{M} \exp(-m_i), \text{ and } \alpha_i = -\frac{\partial L}{\partial m_i}. \tag{2.30}$$

2.3 Methods for solving SVM

In this section, we discuss methods for solving the learning problems. A learning problem is formulated as an optimization problem, and in particular many of them are convex, e.g. the three loss functions discussed in section 2.2 and the L-1 and L-2 norm regularization for linear classifiers are all convex. Therefore general optimization techniques can be applied. However, machine learning poses different requirements for the solution techniques than the studies in mathematical optimization [102]. First, a highly accurate solution is usually not necessary since the training data can be noisy, and the bound on expected loss is probabilistic. Second, fast training and capability to deal with large data is required since usually large amount of training data are available in high dimensional feature space. In this section we study several such methods for training the support vector machines. Two methods that solve the dual SVM are discussed in section 2.3.1 and 2.3.2. The decomposition method [103, 104] in 2.3.1 solves general SVM problems, readily lending it to solving non-linear kernel SVMs. The dual coordinate descent method from [105] is introduced in 2.3.2, and this method

is specialized for linear models without an intercept term. Then two methods that solve the primal problem based on the sub-gradient information are discussed in section 2.3.3 and 2.3.4, i.e. the sub-gradient descent method [49] and the 1-slack cutting plane method [106], both of which are very efficient for linear SVM and can explore very large training set. The n-slack cutting plane method is also discussed in 2.3.4, which constitutes the theoretical basis of retraining with hard examples used in training visual object detectors.

2.3.1 Decomposition methods for dual SVM

We consider solving for a large margin separating hyperplane with arbitrary intercept b:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b. \tag{2.31}$$

The learning problem is:

$$\begin{array}{ll} \min\limits_{\mathbf{w},\boldsymbol{\xi},b} & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum\limits_{i=1}^{M}\xi_i \\ \text{s.t.} & y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i,\, i = 1, ..., M, \\ & \xi_i \geq 0,\, i = 1, ..., M. \end{array} \tag{2.32}$$

Adding the intercept term b induces a slightly different problem as we have discussed in section 2.2.1. The intercept b is not regularized, and the dual problem is:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha}} & W(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{s.t.} & 0 \leq \alpha_i \leq C,\, i = 1, ..., M, \\ & \sum\limits_{i=1}^{M}\alpha_i y_i = 0, \end{array} \tag{2.33}$$

where $Q_{ij} = y_i y_j \mathbf{x}_i^T\mathbf{x}_j$. The dual problem is a quadratic programming problem, with quadratic objective function $W(\boldsymbol{\alpha})$, box constraints $0 \leq \alpha_i \leq C$, and in addition, a linear equality constraint $\sum_{i=1}^{M}\alpha_i y_i = 0$ due to the intercept term b.

The decomposition method solves the dual problem in the following manner, by iterating the following steps:

- Divide the variables in $\{\alpha_i\}$ as the working set B and the fixed set N.
- Solve the dual optimization problem with respect to variables in the working set B.

The method is very similar to coordinate descent for unconstrained problems, but due to the linear equality constraint, the working set B need to contain at least two variables. The method with $|B| = 2$ is called the sequential minimal optimization (SMO) [103] algorithm for its use of a minimal working set B .

After selecting the working set, let $\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{bmatrix}$ and $Q = \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$, we obtain the following sub problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}_B} \quad & \tfrac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B - \boldsymbol{\alpha}_B^T\left(1 - Q_{BN}\boldsymbol{\alpha}_N\right) + const \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, i \in B, \\ & \boldsymbol{\alpha}_B^T \mathbf{y}_B + \boldsymbol{\alpha}_N^T \mathbf{y}_N = 0. \end{aligned} \tag{2.34}$$

This is a small scale problem with only $|B|$ variables, and can be solved efficiently. In the following we discuss two methods for selecting the minimal working set B of two variables.

First order method for working set selection

The first order method [103, 104] is based on the intuition that the selected variables should lead to fastest decrease of the objective value. Therefore the derivative of the objective function is considered as the working set selection criterion. The first order approximation of the dual objective is:

$$\hat{W}\left(\boldsymbol{\alpha} + \mathbf{d}\right) = W\left(\boldsymbol{\alpha}\right) + \mathbf{d}^T g\left(\boldsymbol{\alpha}\right),$$

where $g\left(\boldsymbol{\alpha}\right) = Q\boldsymbol{\alpha} - \mathbf{e}$ is the gradient of $W\left(\boldsymbol{\alpha}\right)$, which can be maintained through the solving process with a small amount of computation. Given the current value of $\boldsymbol{\alpha}$, We seek a increment vector $\mathbf{d}$ to minimize the approximation $\hat{W}\left(\boldsymbol{\alpha} + \mathbf{d}\right)$, and also require that there are only two non-zero elements in $\mathbf{d}$, i.e. the working set size is 2. In the following we denote the index of the two variables as i and j, and use k for general

index of the elements in $\boldsymbol{\alpha}$. The problem is:

$$\begin{aligned} \min_{\mathbf{d}} \quad & \hat{W}(\boldsymbol{\alpha}+\mathbf{d}) = \hat{W}(\boldsymbol{\alpha}) + \mathbf{d}^T g(\boldsymbol{\alpha}) \\ \text{s.t.} \quad & \mathbf{y}^T\mathbf{d} = 0, \\ & d_k \geq 0, \text{ for } k : \alpha_k = 0, \\ & d_k \leq 0, \text{ for } k : \alpha_k = C, \\ & |\{d_k : d_k \neq 0\}| = 2, \\ & \|\mathbf{d}\|_2 = \text{ small const.} \end{aligned} \tag{2.35}$$

The first three constraints come from the feasibility of the solution, and the last constraint is to ensure that the solution does not touch the bounds of $\boldsymbol{\alpha}$. To have a feasible solution for (2.35), all possible choices of the two elements $\{i, j\}$ can be listed as follows:

Variable i	Variable j
$y_i = 1, \alpha_i < C, d_i > 0$	$y_j = 1, \alpha_j > 0, d_j < 0$
$y_i = 1, \alpha_i < C, d_i > 0$	$y_j = -1, \alpha_j < C, d_j > 0$
$y_i = -1, \alpha_i > 0, d_i < 0$	$y_j = 1, \alpha_j > 0, d_j < 0$
$y_i = -1, \alpha_i > 0, d_i < 0$	$y_j = -1, \alpha_j < C, d_j > 0$

Therefore the working set $\{i, j\}$ can be obtained with a simple strategy. Define:

$$\begin{aligned} I_{up}(\boldsymbol{\alpha}) &= \{k | \alpha_k < C, y_k = 1, \text{ or } \alpha_k > 0, y_k = -1\} \\ I_{low}(\boldsymbol{\alpha}) &= \{k | \alpha_k < C, y_k = -1, \text{ or } \alpha_k > 0, y_k = 1\} \end{aligned} \tag{2.36}$$

The choice of the working set is:

$$\begin{aligned} i &= \arg\max_k \{-y_k g_k(\boldsymbol{\alpha}) \,|k \in I_{up}(\boldsymbol{\alpha})\}, \\ j &= \arg\min_k \{-y_k g_k(\boldsymbol{\alpha}) \,|k \in I_{low}(\boldsymbol{\alpha})\} \end{aligned} \tag{2.37}$$

Second order method for working set selection

The second order method [107] considers directly minimizing the objective function rather than its first-order approximation, taking both the working set and the incre-

ments of the variables in the workingset as unknowns:

$$\begin{array}{ll}\min\limits_{\mathbf{d}_B} & W\left(\boldsymbol{\alpha}+\mathbf{d}_B\right)-W\left(\boldsymbol{\alpha}\right)=g_B(\boldsymbol{\alpha})^T\mathbf{d}_B+\frac{1}{2}\mathbf{d}_B^T H_{BB}\left(\boldsymbol{\alpha}\right)\mathbf{d}_B \\ \text{s.t.} & \mathbf{y}_B^T\mathbf{d}_B=0, \\ & d_k\geq 0,\ \text{for}\ k:\alpha_k=0, \\ & d_k\leq 0,\ \text{for}\ k:\alpha_k=C.\end{array} \tag{2.38}$$

$g_B\left(\boldsymbol{\alpha}\right)$ and $H_{BB}\left(\boldsymbol{\alpha}\right)$ are the gradient and Hessian of $W\left(\boldsymbol{\alpha}\right)$ with respect to the variables in the working set B. Since the objective function is quadratic, its second order approximation is exact.

Denote $\hat{d}_i\equiv y_i d_i$ and $\hat{d}_j\equiv y_j d_j$, obviously we have $\hat{d}_i=-\hat{d}_j$ since $\mathbf{y}_B^T\mathbf{d}_B=0$. The objective function becomes:

$$\begin{array}{l}\frac{1}{2}\begin{bmatrix}d_i & d_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ji} & Q_{jj}\end{bmatrix}\begin{bmatrix}d_i\\ d_j\end{bmatrix}+\begin{bmatrix}g_i\left(\boldsymbol{\alpha}\right) & g_j\left(\boldsymbol{\alpha}\right)\end{bmatrix}\begin{bmatrix}d_i\\ d_j\end{bmatrix} \\ =\frac{1}{2}\left(K_{ii}+K_{jj}-2K_{ij}\right)\hat{d}_j^2+\left(-y_i g_i\left(\boldsymbol{\alpha}\right)+y_j g_j\left(\boldsymbol{\alpha}\right)\right)\hat{d}_j\end{array} \tag{2.39}$$

Define $a_{ij}=K_{ii}+K_{jj}-2K_{ij}$, and $b_{ij}=-y_i g_i\left(\boldsymbol{\alpha}\right)+y_j g_j\left(\boldsymbol{\alpha}\right)>0$, the objective function becomes:

$$\frac{1}{2}a_{ij}\hat{d}_j^2+b_{ij}\hat{d}_j,$$

If $a_{ij}>0$, the minimum is at $\hat{d}_j=-\hat{d}_i=-\frac{b_{ij}}{a_{ij}}<0$, and the minimum value is $-\frac{b_{ij}^2}{a_{ij}}<0$, i.e. the objective value is reduced. Therefore if we have selected i, we can select j to directly minimize the objective value. In practice, i is selected using 1-st order information as in [103, 104], and then j is selected to directly minimize the objective function:

$$\begin{array}{l}i=\arg\max\limits_k\left\{-y_k g_k\left(\boldsymbol{\alpha}\right)|k\in I_{up}\left(\boldsymbol{\alpha}\right)\right\} \\ j=\arg\min\limits_k\left\{-\frac{b_{ik}^2}{a_{ik}}|k\in I_{low}\left(\boldsymbol{\alpha}\right),-y_i g_i\left(\boldsymbol{\alpha}\right)+y_k g_k\left(\boldsymbol{\alpha}\right)>0\right\}\end{array} \tag{2.40}$$

If $a_{ij}\leq 0$, which may happen when the kernel is not positive definite, an upper bound of the objective function is minimized:

$$\frac{1}{2}\tau\hat{d}_j^2+b_{ij}\hat{d}_j\geq\frac{1}{2}a_{ij}\hat{d}_j^2+b_{ij}\hat{d}_j,$$

where τ is a small positive value. And the working set selection is as follows:

$$\begin{aligned} i &= \arg\max_k \left\{-y_k g_k(\boldsymbol{\alpha}) \,|k \in I_{up}(\boldsymbol{\alpha})\right\} \\ j &= \arg\min_k \left\{-\tfrac{b_{ik}^2}{\tau} | k \in I_{low}(\boldsymbol{\alpha}), -y_i g_i(\boldsymbol{\alpha}) + y_k g_k(\boldsymbol{\alpha}) > 0\right\} \end{aligned} \tag{2.41}$$

2.3.2 Dual coordinate descent method for linear SVM

The dual coordinate descent algorithm [105] solves a linear SVM without the intercept term:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & y_i\mathbf{w}^T\mathbf{x}_i \geq 1-\xi_i, i = 1, ..., M, \\ & \xi_i \geq 0, i = 1, ..., M. \end{aligned} \tag{2.42}$$

The dual problem is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, ..., M, \end{aligned} \tag{2.43}$$

where $Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$.

By ignoring the intercept term b, the equality constraint is removed from the dual problem, therefore it is possible to modify a single α_i while keeping the solution feasible, resulting in a coordinate descent algorithm. Let $W(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha}$ be the dual objective function. In each iteration of the the coordinate descent algorithm, we select a variable α_i and make a step of size d, which is decided by the following sub problem:

$$\begin{aligned} \min_{d} \quad & W_i(d) \\ \text{s.t.} \quad & 0 \leq \alpha_i + d \leq C, \end{aligned} \tag{2.44}$$

where

$$W_i(d) = \frac{1}{2}Q_{ii}d^2 + g_i(\boldsymbol{\alpha})\, d + const$$

is the objective function that considers only the i-th coordinate α_i. d is the descent step size in this coordinate.

The optimality condition of the dual problem is:

$$\nabla_i^P W(\boldsymbol{\alpha}) = 0, \forall i, \tag{2.45}$$

where $\nabla_i^P W(\boldsymbol{\alpha})$ is the projected gradient:

$$\nabla_i^P W(\boldsymbol{\alpha}) = \begin{cases} \nabla_i W(\boldsymbol{\alpha}) & 0 < \alpha_i < C \\ \min(0, \nabla_i W(\boldsymbol{\alpha})) & \alpha_i = 0 \\ \max(0, \nabla_i W(\boldsymbol{\alpha})) & \alpha_i = C \end{cases} \tag{2.46}$$

Therefore given the selected coordinate i, the step size d is obtained by setting the derivative to 0:

$$\frac{\partial W_i(\alpha_i + d)}{\partial d} = Q_{ii}d + g_i(\boldsymbol{\alpha}) = 0 \Rightarrow d = -\frac{g_i(\boldsymbol{\alpha})}{Q_{ii}}. \tag{2.47}$$

The update in the coordinate is:

$$\alpha_i^* \leftarrow \alpha_i + d = \alpha_i - \frac{g_i(\boldsymbol{\alpha})}{Q_{ii}}, \tag{2.48}$$

and we project it back to the feasible set of α_i:

$$\alpha_i^* \leftarrow \min\left(\max\left(\alpha_i - \frac{g_i(\boldsymbol{\alpha})}{Q_{ii}}, 0\right), C\right). \tag{2.49}$$

If Q_{ii} is 0, the solution is simply $\alpha_i = C$.

For linear SVM, the primal variable $\mathbf{w}$ can be updated with each coordinate descent step, and the gradient of the dual objective with respect to each dual variable can be computed as follows:

$$g_i(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1 = y_i \mathbf{w}^T \mathbf{x}_i - 1$$

Therefore each coordinate descent is very cheap, and the dual coordinate descent algorithm is very efficient for solving linear SVM without the intercept term.

2.3.3 Primal sub-gradient descent method for linear SVM

The primal sub-gradient descent algorithm [49] solves the linear SVM without the intercept term b, using the following unconstrained problem formulation:

$$\min_{\mathbf{w}} \quad \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{M}\sum_{i=1}^{M} \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) \tag{2.50}$$

The objective function is convex but non-differentiable. We can calculate a sub-gradient of the objective function at location $\mathbf{w}_t$:

$$\nabla_t = \lambda \mathbf{w}_t - \frac{1}{M}\sum_{i=1}^{M} \pi_{t,i} y_i \mathbf{x}_i, \text{ where } \pi_{t,i} = \begin{cases} 1 & y_i \mathbf{w}_t^T \mathbf{x}_i \leq 1 \\ 0 & y_i \mathbf{w}_t^T \mathbf{x}_i > 1 \end{cases} \tag{2.51}$$

Instead of using the full training set to calculate the sub-gradient, a much smaller set A_t of training instances can be used to approximate the sub-gradient:

$$\nabla_t \approx \lambda \mathbf{w}_t - \frac{1}{|A_t|}\sum_{i \in A_t} \pi_{t,i} y_i \mathbf{x}_i \tag{2.52}$$

In the extreme case, we approximate the sub-gradient using one single selected training instance i, i.e. $\nabla_t = \lambda \mathbf{w}_t - \pi_{t,i} y_i \mathbf{x}_i$. We can sequentially loop over the training instances, or randomly pick a training instance in each step. After calculating the sub-gradient, a step size $\eta_t = \eta_0/t$ is performed in the negative sub-gradient direction:

$$\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_t = (1 - \eta_t \lambda)\, \mathbf{w}_t + \eta_t \frac{1}{|A_t|}\sum_{i \in A_t} \pi_{t,i} y_i \mathbf{x}_i \tag{2.53}$$

Furthermore, the dual problem can be written as:

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha}} & -\frac{1}{2\lambda}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \sum\limits_{i=1}^{M} \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \leq 1/M. \end{array} \tag{2.54}$$

The KKT conditions require that $\mathbf{w} = \frac{1}{\lambda}\sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i$, therefore at the optimum, the dual objective is equal to $-\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \mathbf{e}^T\boldsymbol{\alpha}$. The primal objective should be equal to the dual objective at optimal, i.e.:

$$\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{M}\sum_{i=1}^{M} \xi_i = -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \mathbf{e}^T\boldsymbol{\alpha} \tag{2.55}$$

Therefore we can obtain:

$$\lambda \mathbf{w}^T\mathbf{w} = \mathbf{e}^T\boldsymbol{\alpha} - \frac{1}{M}\sum_{i=1}^{M} \xi_i \leq 1 \Rightarrow \|\mathbf{w}\|_2 \leq \sqrt{1/\lambda} \tag{2.56}$$

which indicates the optimal solution is inside a ball of radius $\sqrt{1/\lambda}$. Therefore in the sub-gradient descent method, we project $\mathbf{w}_{t+\frac{1}{2}}$ back to inside the ball $\|\mathbf{w}\|_2 \leq \sqrt{1/\lambda}$

to obtain:

$$\mathbf{w}_{t+1} = \mathbf{w}_{t+1/2} \min\left(1, \frac{1}{\left\|\mathbf{w}_{t+1/2}\right\|_2 \sqrt{\lambda}}\right). \tag{2.57}$$

This shows improved convergence speed in practice.

2.3.4 Cutting plane methods

n-slack cutting plane method: Exploring the hard examples

First we discuss the n-slack cutting plane method, which solves the following problem:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & y_i\mathbf{w}^T\mathbf{x}_i \geq 1-\xi_i,\, i = 1, ..., M, \\ & \xi_i \geq 0,\, i = 1, ..., M. \end{aligned} \tag{2.58}$$

The intercept term b can also be incorporated without affecting the following discussion. This problem has M linear inequality constraints for $\mathbf{w}$, which defines the feasible set of $\mathbf{w}$. The cutting plane method for inequality constrained problems approximates the feasible set using a number of cutting planes. A cutting plane is defined at a position $\mathbf{w}_t$, and corresponds to an inequality constraint in the original problem.

We use function $g_i(\mathbf{w}) = 1 - \xi_i - y_i\mathbf{w}^T\mathbf{x}_i \leq 0$ to represent the i-th constraint on $\mathbf{w}$ in problem (2.58). A cutting plane is added for $g_i(\mathbf{w})$ at position $\mathbf{w}_t$ as:

$$\hat{g}_{i,t}(\mathbf{w}) = g_i(\mathbf{w}_t) + (\mathbf{w}-\mathbf{w}_t)^T \left.\frac{\partial g_i(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t} \leq 0 \tag{2.59}$$

The cutting plane $\hat{g}_{i,t}(\mathbf{w})$ is a under estimator of $g_i(\mathbf{w})$ since $g_i(\mathbf{w})$ is convex. In particular, for the linear constraints, the cutting plane is the function itself, i.e. $\hat{g}_{i,t}(\mathbf{w}) = g_i(\mathbf{w})$. The cutting plane method for constrained optimization problem can be described as follows.

Starting with a relaxed problem without any constraint $g_i(\mathbf{w})$, i.e. without any training instance. The initial feasible set of $\mathbf{w}$ is the full space. Then in each iteration, we get a solution $\mathbf{w}_t$ for the relaxed problem, and add cutting planes for those violated constraints $g_i(\mathbf{w}) > 0$, i.e. including those hard instances into the training set. The effect is that part of the feasible set of $\mathbf{w}$ is cut off by adding the constraints. Those inactive cutting planes $g_i(\mathbf{w}) < 0$, i.e. the easy training instances, can be removed from the problem without affecting the solution. Therefore the procedure is:

- Initialize the solution $\mathbf{w}_0 = 0$ and $\xi_i = 0$, $\forall i$, and the training set $S_0 = \varnothing$.
- Iterate:
  - Update the training set by collecting the in-margin instances from the full training set, ie. $\{i : g_i(\mathbf{w}_t) > 0\}$, and remove those easy training instances.
  - Train the classifier using the new training set S_t.

Therefore SVMs can be efficiently trained using the n-slack cutting plane algorithm for very large training datasets. Instead of directly training on the full dataset, we train the classifier using a small subset, and then update the working set of training instances by removing those easy instances, and adding those hard instances. The solution converges to the result of training on the full dataset. An example of using the n-slack cutting plane algorithm for training a linear SVM classifier using a negative training set comprising all windows in the negative images (i.e. images not containing any instance of the object category) is shown in Table 2.1. We constrain the working set of negative instances to be at most 10,000. The first working set is randomly taken from the negative images. Then the working set is updated by keeping the hard negatives in the current working set, and including those most difficult negative windows in the training images. The table shows the number of hard negative windows (SVM score ≥ -1) in the negative training images with each trained classifier. It can be observed that the process quickly identifies those hard negatives, which comprise less than 0.1% of the total windows in the negative images. The small set of most difficult negative instances can fit into memory, and a classifier trained using these hard negatives is exactly identical to that trained using the full negative dataset with multi millions of training instances.

Table 2.1: Number of hard negative training instances in a n-slack cutting plane training process.

Iteration	0	1	2	3	4
# hard neg	11,801,876	704,225	250,029	9,364	5,898

1-slack cutting plane method for linear SVM

The 1-slack cutting plane method [106] considers the following unconstrained optimization problem:

$$\min_{\mathbf{w}} \quad \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{M}\sum_{i=1}^{M} \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right) \tag{2.60}$$

The problem is convex, but the empirical loss $L(\mathbf{w}) = \frac{1}{M}\sum_{i=1}^{M} \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$ is not differentiable everywhere. Using the sub-gradient, the cutting planes of L at positions $\mathbf{w}_t$ is a linear function:

$$\hat{L}_t(\mathbf{w}) = L(\mathbf{w}_t) + (\mathbf{w} - \mathbf{w}_t)^T g(\mathbf{w}_t), \tag{2.61}$$

where $g(\mathbf{w}_t)$ is the sub-gradient of $L(\mathbf{w})$ at $\mathbf{w}_t$. Then the loss function is bounded from below by the pointwise maximum of the cutting planes:

$$L(\mathbf{w}) \geq \max_t \left\{\hat{L}_t(\mathbf{w})\right\}. \tag{2.62}$$

The bound is tight if sufficiently many cutting planes are constructed in the proper manner. Furthermore, $\hat{L}_0(\mathbf{w}) = 0$ is a cutting plane of $L(\mathbf{w})$ at $\mathbf{w} = 0$. Therefore

$$L(\mathbf{w}) \geq L_T^* = \max\left(0, \max_{t=1,\ldots,T}\left\{\hat{L}_t(\mathbf{w})\right\}\right). \tag{2.63}$$

By progressively adding cutting planes to form tighter and tighter lower bounds of the original objective function (2.60), the cutting plane method for solving the unconstrained convex optimization problem (2.60) works as follows:

- Initialize the solution to be $\mathbf{w}_0 = 0$.
- For $T = 0, ..., T_{MAX}$
  - Add a cutting plane $\hat{L}_T(\mathbf{w})$ at the current solution $\mathbf{w}_T$.
  - Approximate the loss function using L_T^*, i.e. the pointwise maximum of the cutting planes, and then solve the problem to obtain a solution $\mathbf{w}_{T+1}$.

For the SVM problem, a sub-gradient of the loss function is:

$$g\left(\mathbf{w}_t\right) = -\frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}y_i\mathbf{x}_i, \text{ where } \pi_{t,i} = \begin{cases} 1 & y_i\mathbf{w}_t^T\mathbf{x}_i \leq 1 \\ 0 & y_i\mathbf{w}_t^T\mathbf{x}_i > 1 \end{cases} \tag{2.64}$$

The indicator $\pi_{t,i}$ selects those hard (in-margin and on-margin) instances using the current solution $\mathbf{w}_t$. The cutting plane is:

$$\begin{aligned} \hat{L}_t\left(\mathbf{w}\right) &= L\left(\mathbf{w}_t\right) + \left(\mathbf{w}-\mathbf{w}_t\right)^T g\left(\mathbf{w}_t\right) \\ &= \frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}\left(1-y_i\mathbf{x}_i\right) \\ &= \frac{1}{M}\sum_{i=1}^{M}\pi_{t,i} - \mathbf{w}^T\frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}y_i\mathbf{x}_i \end{aligned} \tag{2.65}$$

And the approximated problem is:

$$\min_{\mathbf{w}} \quad \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \max\left(0, \max_{t=1,\ldots,T}\left\{\frac{1}{M}\sum_{i=1}^{M}\pi_{t,i} - \mathbf{w}^T\frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}y_i\mathbf{x}_i\right\}\right), \tag{2.66}$$

which is equivalent to:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \xi \\ \text{s.t.} \quad & \mathbf{w}^T\frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}y_i\mathbf{x}_i \geq \frac{1}{M}\sum_{i=1}^{M}\pi_{t,i} - \xi, t = 1, \ldots, T, \\ & \xi \geq 0. \end{aligned} \tag{2.67}$$

Therefore the cutting plane algorithm for the unconstrained problem is equivalent to the cutting plane algorithm that cuts the feasible set of $\mathbf{w}$ by sequentially adding constraints for the following problem:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \xi \\ \text{s.t.} \quad & \mathbf{w}^T\frac{1}{M}\sum_{i=1}^{M}\pi_i y_i\mathbf{x}_i \geq \frac{1}{M}\sum_{i=1}^{M}\pi_i - \xi, \forall\boldsymbol{\pi} \in \{0,1\}^M, \\ & \xi \geq 0, \end{aligned} \tag{2.68}$$

since for any current solution $\mathbf{w}_t$, $\pi_{t,i} = \begin{cases} 1 & y_i\mathbf{w}_t^T\mathbf{x}_i \leq 1 \\ 0 & y_i\mathbf{w}_t^T\mathbf{x}_i > 1 \end{cases}$ is also the labeling that maximizes $\frac{1}{M}\sum_{i=1}^{M}\pi_i - \mathbf{w}^T\frac{1}{M}\sum_{i=1}^{M}\pi_i y_i\mathbf{x}_i$, i.e. the mostly violated constraint in the problem (2.68).

Compared with the original SVM problem, (2.68) introduces hypothetical training instances with actual margin $\frac{1}{M}\sum_{i=1}^{M}\pi_i y_i \mathbf{x}_i$, and target margin $\frac{1}{M}\sum_{i=1}^{M}\pi_i$. Furthermore, only one slack variable ξ is required, and therefore (2.68) is called the 1-slack formulation. It can be verified that the solution of 1-slack formulation is equal to that of the n-slack formulation, since the following holds for any each $\mathbf{w}$:

$$\begin{aligned}\xi &= \max_{\boldsymbol{\pi}} \left\{ \frac{1}{M}\sum_{i=1}^{M}\pi_i - \mathbf{w}^T \frac{1}{M}\sum_{i=1}^{M}\pi_i y_i \mathbf{x}_i \right\} \\ &= \frac{1}{M}\sum_{i=1}^{M} \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) \\ &= \frac{1}{M}\sum_{i=1}^{M}\xi_i \end{aligned} \tag{2.69}$$

Therefore the optimal value of (2.68) is identical to the optimal value of (2.60), and the solution of the 1-slack formulation is identical to that of the n-slack formulation. Next we derive the dual problem for the 1-slack problem (2.67). Let $\Psi_t = \frac{1}{M}\sum_{i=1}^{M}\pi_{t,i} y_i \mathbf{x}_i$, and $\Delta_t = \frac{1}{M}\sum_{i=1}^{M}\pi_{t,i}$, (2.67) can be written as:

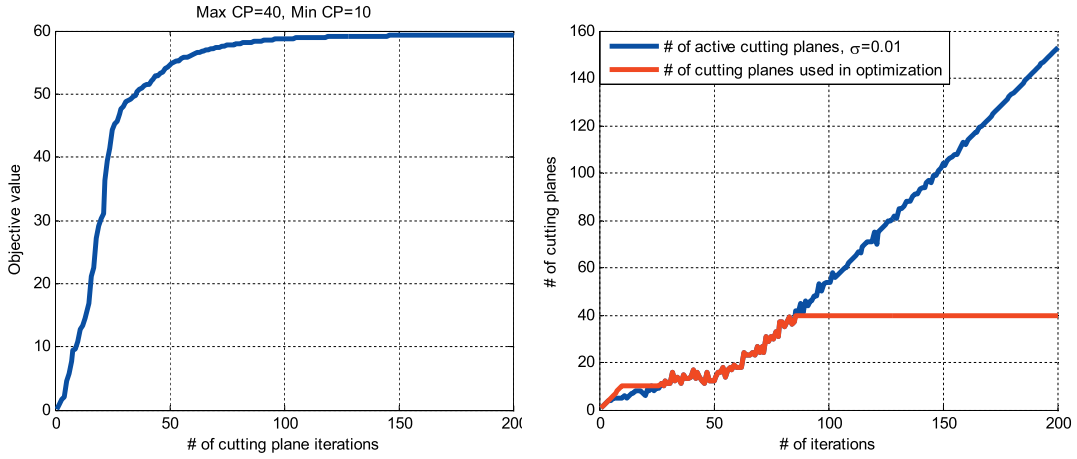
$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \xi \\ \text{s.t.} \quad & \mathbf{w}^T\Psi_t \geq \Delta_t - \xi, t = 1, ..., T, \\ & \xi \geq 0. \end{aligned} \tag{2.70}$$

The Lagrangian is:

$$\mathcal{L}\left(\mathbf{w}, \xi, \boldsymbol{\alpha}, \beta\right) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \xi + \sum_{t=1}^{T}\alpha_i\left(\Delta_t - \xi - \mathbf{w}^T\Psi_t\right) - \beta\xi \tag{2.71}$$

The KKT conditions are:

Stationary	Primal and dual feasibility	Complementary slackness
$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \lambda\mathbf{w} - \sum_{t=1}^{T}\alpha_t\Psi_t = 0$	$\mathbf{w}^T\Psi_t \geq \Delta_t - \xi, \forall t$	$\alpha_t\left(\Delta_t - \xi - \mathbf{w}^T\Psi_t\right) = 0$
$\frac{\partial\mathcal{L}}{\partial\xi} = 1 - \sum_{t=1}^{T}\alpha_t - \beta = 0$	$\xi \geq 0$	$\beta\xi = 0$
	$\alpha_i \geq 0$	
	$\beta \geq 0$	



(a) (b)

Figure 2.2: An example of 1-slack cutting plane training of linear SVM. (a) The objective function converges from below as more cutting planes are added. (b) By using only the most difficult cutting planes, the complexity of each subproblem is effectively constrained without impairing the quality of the solution.

The dual problem is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \tfrac{1}{2\lambda}\boldsymbol{\alpha}^T\hat{Q}\boldsymbol{\alpha} - \boldsymbol{\Delta}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_{t=1}^{T}\alpha_t \leq 1, \\ & \alpha_t \geq 0, t = 1, ..., T. \end{aligned} \tag{2.72}$$

where $\hat{Q}$ is a $T \times T$ matrix, $\hat{Q}_{ij} = \Psi_i^T\Psi_j$. There are T variables in the dual problem, and usually the number of cutting planes needed to obtain a good solution is significantly smaller than the training set size M. Therefore the complexity of 1-slack training scales approximately as $O(M)$ respect to the dataset size, and is a very efficient method for linear SVM. Furthermore, in the later iterations, the in-active cutting planes, i.e. $\mathbf{w}_T^T\Psi_t \geq \Delta_t - \xi + \sigma$, where $\mathbf{w}_T$ is the current solution, and σ is a small positive constant, can be removed from the problem without affecting the solution. We also specify a minimum number n_{min} and a maximum number n_{max} of cutting planes used in solving problem (2.70). If the number of active cutting plane is smaller than n_{min} or larger than n_{max}, we keep the cutting planes with the top n_{min} or n_{max} highest values of $\Delta_t - \mathbf{w}_T^T\Psi_t$ in problem (2.70) and (2.72). n_{min} prevents the solution drifting and helps convergence, since an in-active cutting plane at $\mathbf{w}_T$ could be active for the next solution $\mathbf{w}_{T+1}$. n_{max} keeps the problem (2.70) and (2.72) at manageable size. Fig. 2.2a shows the convergence of the objective function. The cutting plane algorithm approaches the optimal value from below as more cutting planes are intro-

duced. Fig. 2.2b shows how the problem size grows and is controlled by $n_{min} = 10$ and $n_{max} = 40$.

2.4 Embedded forward feature selection

In this section, we derive an embedded forward feature selection method for a class of structural risk minimization learning problems. The feature selection problem is described as follows.

Denote the features we use to represent a sample instance as a feature set H, which comprises a number of feature channels, i.e. $H = H_1 \bigcup ... \bigcup H_P$. The full feature space can be represented by $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_P$. The feature vector for a sample instance can be obtained by the concatenation of features in H, i.e. $\mathbf{x} \in \mathcal{H}$. We want to select a feature subset $\hat{H} \subseteq H$, such that the classifier using this feature subset has good performance. The feature selection is of interest for several reasons. First, it produces efficient classifiers since only those selected features need to be extracted at testing time. Second, it results in easier learning problems since we are working with a low dimensional feature space $\hat{\mathcal{H}}$ limited to those selected features $\hat{H}$. Third, the selected features often provide useful insight into the nature of the problem.

Methods for feature selection can be divided into three categories [56], the filter methods, the wrapper methods, and the embedded methods. The filter methods assign an importance score to each feature, and then select features according to their importance scores. Feature selection and learning are two independent stages in this case. The wrapper methods treat the classifier as a black box, and test subsets of features by training a particular classifier using these features. Since exhaustively enumerating all subsets is impractical, greedy strategies are usually used instead. The forward feature selection starts with an empty set $\varnothing$, and adds the feature that mostly improves the performance in each iteration. The backward feature selection starts with the full set H, and removes the feature that least affects the performance in each iteration. For the embedded methods, the feature selection is performed along with training the classifier, and information from the trained classifier is exploited for finding the good features. The embedded methods are of most interest for two reasons. First, they usually have higher efficiency over the wrapper methods, since information of the classifier can be exploited to guide the feature selection, rather than retraining the classifier using the modified feature set. Second, often better feature subsets can be produced than that by the filtering methods, since the features are selected to optimize the classifier's

performance, and can be expected to be complementary to each other, while the high-scored features in filtering methods are usually highly correlated with each other. In this section, we introduce an embedded forward feature selection method for a class of structural risk minimization learning problems.

2.4.1 A forward feature selection algorithm

Column generation method for feature selection

We consider $\mathbf{x} \in \mathbb{R}^N$, and linear model $f = \mathbf{w}^T\mathbf{x}$, and consider the class of loss functions separable in the training instances, i.e.:

$$L(\mathbf{w}) = \sum_{i=1}^{M} l\left(y_i, \mathbf{w}^T\mathbf{x}_i\right). \tag{2.73}$$

By introducing the auxiliary variables $f_i = \mathbf{w}^T\mathbf{x}_i$, and a regularization term $R(\mathbf{w})$, the learning problem becomes:

$$\begin{array}{ll} \min\limits_{\mathbf{f},\mathbf{w}} & \lambda R(\mathbf{w}) + \sum\limits_{i=1}^{M} l(y_i, f_i) \\ \text{s.t.} & f_i = \mathbf{w}^T\mathbf{x}_i,\, i = 1, ..., M. \end{array} \tag{2.74}$$

We assume the domain of $\mathbf{w}$ is $\mathbf{w} \in \mathbb{R}^N$. To see how the features can be introduced one by one, we write the Wolfe dual of the problem:

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha},\mathbf{f},\mathbf{w}} & \lambda R(\mathbf{w}) + \sum\limits_{i=1}^{M} l(y_i, f_i) + \sum\limits_{i=1}^{M} \alpha_i \left(f_i - \mathbf{w}^T\mathbf{x}_i\right) \\ s.t. & f_i = \mathbf{w}^T\mathbf{x}_i,\, \frac{\partial l(y_i,f_i)}{\partial f_i} + \alpha_i = 0,\, i = 1, ..., M, \\ & \lambda\frac{\partial R}{\partial w_j} - \sum\limits_{i=1}^{M} \alpha_i x_{ij} = 0,\ j = 1, ..., N. \end{array} \tag{2.75}$$

If $R(\mathbf{w})$ and $l(y_i, f_i)$ are both convex, the optimal values of the primal and dual problems are equal. For the dual problem, it can be observed that each feature j now correspond to a constraint $\lambda\frac{\partial R}{\partial w_j} - \sum_{i=1}^{M} \alpha_i x_{ij} = 0$. Selecting a subset of features is equivalent to fixing some w_j to 0 and ignoring the corresponding constraints, i.e. relaxing the dual problem. Therefore a forward feature selection method can be derived from the cutting plane algorithm, which relaxes the problem (2.75) and sequentially add back the constraints, i.e. performs forward feature selection. For this problem, the

cutting plane algorithm is also known as the column generation algorithm. Denote matrix H in which each row $H_i = \mathbf{x}_i$ is the feature vector of a training instance and each column $H_{:j}$ is the j-th feature for all training instances. In the following we use H for both the matrix of features and the feature set used to represent the training instances. The column generation method sequentially generates the columns of the matrix of selected features $\hat{H}$. The algorithm solves a sequence of relaxed dual problems, each being tighter (with more constraints, i.e. more features) than the previous one:

- Start with $\hat{H} = \varnothing$, iterate the following steps:
  - Solve the relaxed dual problem with $\hat{H}$, i.e. represent the training instances using the selected feature subset. This is equivalent to ignoring the constraints $j \notin \hat{H}$, and fixing the corresponding w_j to 0. Obviously $w_j = 0$ is in the feasible set of $\mathbf{w}$, therefore the solution of the relaxed problem will always be feasible for the original problem.
  - Find the maximally violated dual constraint:

$$\arg\max_{j \notin \hat{H}} \left| \lambda \frac{\partial R}{\partial w_j} - \sum_{i=1}^{M} \alpha_i x_{ij} \right|,$$

    and update the selected feature set: $\hat{H} \leftarrow \hat{H} \bigcup \{j\}$. Note that if the loss function is differentiable, the dual variables $\{\alpha_i\}$ can be obtained in closed form as $\alpha_i = -\frac{\partial l(y_i, f_i)}{\partial f_i}$. Otherwise the problem needs to be modified, for which we will show an example later.

In the following we show the implication of adding the most violated constraint from two aspects, i.e. optimizing the Lagrangian, and coordinate descent of the unconstrained primal problem.

Lagrangian perspective of the column generation method

For a constrained convex optimization problem:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \\ & h_j(x) = 0. \end{aligned} \tag{2.76}$$

The Lagrangian is:

$$\mathcal{L} = f(x) + \sum_i \alpha_i g_i(x) + \sum_j \beta_j h_j(x). \tag{2.77}$$

The most violated constraint corresponds to the dual variable that has the largest absolute derivative:

$$\begin{aligned} \arg\max_{i,g_i(x)>0} g_i(x) &= \arg\max_{i,g_i(x)>0} \left|\frac{\partial \mathcal{L}}{\partial \alpha_i}\right| \\ \arg\max_{j,h_j(x)\neq 0} |h_j(x)| &= \arg\max_{j,h_j(x)\neq 0} \left|\frac{\partial \mathcal{L}}{\partial \beta_j}\right| \end{aligned} \tag{2.78}$$

The Lagrangian (2.77) is a lower bound of the original problem (2.76). The column generation method is essentially a coordinate ascent method of the Lagrangian, starting with $\boldsymbol{\alpha} = 0$ and $\boldsymbol{\beta} = 0$, and in each step selecting a coordinate α_i or β_j to modify according to the first order derivative, and forming tighter lower bounds of the original problem. For the Wolfe dual (2.75), since it is a maximization problem, the column generation method approaches the optimal value from above, such that the original objective (2.74) is reduced with the inclusion of new features.

Primal coordinate descent perspective of the column generation method

The column generation criterion is equivalent to the 1-st order criterion for selecting coordinates in a coordinate descent method for solving the following unconstrained problem:

$$\min_{\mathbf{w}} \quad \lambda R(\mathbf{w}) + \sum_{i=1}^{M} l\left(y_i, \mathbf{w}^T \mathbf{x}_i\right). \tag{2.79}$$

Starting with $\mathbf{w} = 0$, the coordinate descent method selects one w_j in each iteration. The first order derivative can be used for selecting the coordinate:

$$\arg\max_j \left| \lambda \frac{\partial R}{\partial w_j} + \sum_{i=1}^{M} \frac{\partial l\left(y_i, \mathbf{w}^T \mathbf{x}_i\right)}{\partial w_j} \right|. \tag{2.80}$$

Then a step is performed in this direction by: $w_j \leftarrow w_j + \eta$. The step size can be decided by optimization, or using a shrinking strategy: $\eta_t = \eta_0 / t$.

For the previous column generation method, if the loss l is differentiable, the α_i can be obtained in closed form as $\alpha_i = -\partial l(y_i, f_i)/\partial f_i$, and the column generation

criterion is equivalent to finding the coordinate with maximum derivative, since:

$$\sum_{i=1}^{M} \frac{\partial l\left(y_i, \mathbf{w}^T\mathbf{x}_i\right)}{\partial w_j} = \sum_{i=1}^{M} \frac{\partial l\left(y_i, f_i\right)}{\partial f_i}\frac{\partial f_i}{\partial w_j} = -\sum_{i=1}^{M} \alpha_i x_{ij} \tag{2.81}$$

If l is not differentiable, e.g. the hinge loss, we can use its arbitrary sub-gradient $\partial l\left(y_i, f_i\right)/\partial f_i$, since the objective value is assured to decrease along any sub-gradient direction. Or we may solve a modified dual problem to obtain α_i, which will be discussed in section 2.4.2. Therefore the forward feature selection algorithm can be implemented for general problems by the following procedure:

- Solve the learning problem with the reduced feature set $\hat{H}$.
- Select new features into $\hat{H}$ according to the gradient/sub-gradient of the features (2.80).

2.4.2 Special cases

Non-differentiable loss: The Hinge loss

For the L-2 regularized SVM problem:

$$l\left(y_i, \mathbf{w}^T\mathbf{x}_i\right) = \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right), \text{ and } R\left(\mathbf{w}\right) = \mathbf{w}^T\mathbf{w}. \tag{2.82}$$

The loss is not differentiable. Introducing the slack variables $\{\xi_i\}$, the learning problem is:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{f},\boldsymbol{\xi}} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & f_i = \mathbf{w}^T\mathbf{x}_i, \\ & y_i f_i \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{2.83}$$

Its Wolfe dual is:

$$\begin{array}{ll}\max\limits_{\mathbf{w},\mathbf{f},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}} & \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}+\sum\limits_{i=1}^{M}\xi_i+\sum\limits_{i=1}^{M}\alpha_i\left(f_i-\mathbf{w}^T\mathbf{x}_i\right)+\sum\limits_{i=1}^{M}\beta_i\left(1-\xi_i-y_if_i\right)-\sum\limits_{i=1}^{M}\gamma_i\xi_i \\ \text{s.t.} & f_i=\mathbf{w}^T\mathbf{x}_i, \\ & \beta_i\geq 0,\gamma_i\geq 0, \\ & \alpha_i-\beta_iy_i=0,1-\beta_i-\gamma_i=0, \\ & \lambda w_j-\sum\limits_{i=1}^{M}\alpha_ix_{ij}=0.\end{array} \tag{2.84}$$

Therefore the column generation criterion is:

$$\arg\max_j\left|\lambda w_j-\sum_{i=1}^{M}\alpha_ix_{ij}\right|, \tag{2.85}$$

The dual variable $\boldsymbol{\alpha}$ can be obtained from the Lagrangian dual problem (2.10). According to the analysis in section 2.2.1, it is easy to verify that $-\alpha_i$ is a sub-gradient of $l\left(y_i,f_i\right)$, since:

$$\frac{\partial l\left(y_i,f_i\right)}{\partial f_i}=\begin{cases}0, & y_i\mathbf{w}^T\mathbf{x}_i>1 \\ \text{any value in }\left[0,-y_i\right]\bigcup\left[-y_i,0\right], & y_i\mathbf{w}^T\mathbf{x}_i=1 \\ -y_i, & y_i\mathbf{w}^T\mathbf{x}_i<1\end{cases} \tag{2.86}$$

and we have:

$$\begin{array}{lll}\alpha_i=\beta_iy_i, & \beta_i=0 & \text{if } y_if_i>1 \\ & 0\leq\beta_i\leq 1 & \text{if } y_if_i=1 \\ & \beta_i=1 & \text{if } y_if_i<1\end{array} \tag{2.87}$$

Non-differentiable regularizer: L-1 norm regularization

The regularization term $R\left(\mathbf{w}\right)$ may not be differentiable. In this section, we study the L-1 norm regularization $R\left(\mathbf{w}\right)=\sum\limits_{j=1}^{M}|w_j|$.

We generate an extended feature vector $\mathbf{x}^*=\begin{bmatrix}\mathbf{x}\\-\mathbf{x}\end{bmatrix}$. In the following we use the superscript * to refer to working with the extended feature vector $\mathbf{x}^*$. Then the L-1 regularized problem can be converted to the following problem with a differentiable

regularizer, but constraining the domain of $\mathbf{w}^*$ to be nonnegative:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{f}} \quad & \lambda \sum_{j=1}^{2N} w_j^* + \sum_{i=1}^{M} L\left(y_i, f_i\right) \\ \text{s.t.} \quad & f_i = \mathbf{w}^{*T}\mathbf{x}_i^*, \\ & w_j^* \geq 0. \end{aligned} \tag{2.88}$$

Its Wolfe dual is:

$$\begin{aligned} \max_{\mathbf{w},\mathbf{f},\boldsymbol{\alpha},\boldsymbol{\beta}} \quad & \lambda \sum_{j=1}^{2N} w_j^* + \sum_{i=1}^{M} L\left(y_i, f_i\right) + \sum_{i=1}^{M} \alpha_i \left(f_i - \mathbf{w}^T\mathbf{x}_i\right) - \sum_{j=1}^{2N} \beta_j w_j^* \\ \text{s.t.} \quad & f_i = \mathbf{w}^T\mathbf{x}_i, w_j^* \geq 0, \\ & \beta_i \geq 0, \\ & \lambda - \sum_{i=1}^{M} \alpha_i x_{ij} - \beta_j = 0, \frac{\partial L(y_i,f_i)}{\partial f_i} + \alpha_i = 0. \end{aligned} \tag{2.89}$$

Let $J^* = \left\{j \in \{1, \cdots, 2N\} \setminus \hat{H}^* : \sum_{i=1}^{M} \alpha_i x_{ij} > \lambda\right\}$, then the column generation criterion is:

$$\arg\min_{j \in J^*} \left(\lambda - \sum_{i=1}^{M} \alpha_i x_{ij}\right), \text{i.e. } \arg\max_{j \in J^*} \sum_{i=1}^{M} \alpha_i x_{ij}, \tag{2.90}$$

which is essentially identical to $\arg\max_{j \in J} \sum_{i=1}^{M} \alpha_i x_{ij}$, where

$$J = \left\{j \in \{1, \cdots, N\} \setminus \hat{H}, \left|\sum_{i=1}^{M} \alpha_i x_{ij}\right| > \lambda\right\},$$

and with a sign reverse post processing: $j^* = j + N$ if $\sum_{i=1}^{M} \alpha_i x_{ij} < 0$, i.e. use the oppositely signed feature to ensure that a positive step in the negative gradient direction improves the objective value, i.e. to ensure w_j^* is non-negative. The criterion is identical to using the projected gradient for the following problem:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{f}} \quad & \lambda \sum_{j=1}^{2N} w_j^* + \sum_{i=1}^{M} l\left(y_i, \mathbf{w}^{*T}\mathbf{x}_i^*\right) \\ \text{s.t.} \quad & w_j^* \geq 0. \end{aligned} \tag{2.91}$$

By projecting the gradient of the objective function onto the feasible set of $\mathbf{w}$, the

projected gradient of the objective function is:

$$\begin{cases} \lambda + \sum_{i=1}^{M} \frac{\partial l(y_i, f_i)}{\partial f_i} x_{ij}^* & w_j^* > 0 \\ \min\left(0, \lambda + \sum_{i=1}^{M} \frac{\partial l(y_i, f_i)}{\partial f_i} x_{ij}^*\right) & w_j^* = 0 \end{cases} \tag{2.92}$$

The column generation criterion is also equivalent to using sub-gradient of the following unconstrained primal problem:

$$\min_{\mathbf{w}} \quad \lambda \sum_{j=1}^{N} |w_j| + \sum_{i=1}^{M} l\left(y_i, \mathbf{w}^T \mathbf{x}_i\right) \tag{2.93}$$

The sub differential of $r(w) = |w|$ evaluated at $w = 0$ is $[-\lambda, \lambda]$, therefore a sub-gradient of the objective function at $w_j = 0$ is:

$$\sum_{i=1}^{M} \frac{\partial l(y_i, f_i)}{\partial f_i} \frac{\partial f_i}{\partial w_j}. \tag{2.94}$$

The column generation criterion finds the feature to maximize the magnitude of this sub-gradient in each iteration. Furthermore, if $\max_{j \in J} \left| \sum_{i=1}^{M} \alpha_i x_{ij} \right| \leq \lambda$, then no constraint is violated in (2.89), all projected gradient in (2.92) are 0, and in (2.93) moving in any negative sub-gradient direction will increase the objective value, therefore no more features will be selected. This property ensures that the L-1 regularization is inherently able to perform feature selection even without the forward feature selection method, with the sparsity controlled implicitly by λ.

2.4.3 Experimental evaluation

To illustrate the forward feature selection method, we design a classification task on the INRIA pedestrian dataset. 10,000 negative instances are collected for training, and another 10,000 for testing, both of which are collected from the hard windows (score of a linear SVM + HOG detector ≥ -1) from the negative training and testing images, respectively. 2,474 and 1,178 positive instances are used for training and testing, respectively. A HOG feature vector is extracted from 105 size 16×16 blocks and 21 size 32×32 blocks in the cropped object window of size 128×64 pixels. A length 32 histogram is used to represent each block, and therefore the feature vector length is 4,032.

Three methods for feature selection are compared. The first method is the recursive

feature elimination (RFE) method from [56], which is a backward elimination method. Starting with the full feature set, it trains a linear SVM classifier, and then treats the SVM parameter $\mathbf{w}$ as weights for the features. Then an elimination rule is applied based on $\mathbf{w}$. For example, we remove those lowest weighted features comprising the 2% of the total weights in each iteration. The second method is based on the L-1 regularization, which implicitly introduces sparsity in the resulting $\mathbf{w}$, and the sparsity is controlled by the regularization parameter λ in (2.93). The last method is our proposed forward feature selection (FFS) method described in section 2.4.1. The L-2 regularized SVM and L-1 regularized SVM problems are solved using the liblinear package [108].

We show the ROC curves on the testing set using various number of features selected by the FFS algorithm in Fig. 2.3a. It can be observed that the using a fraction of the full feature set (e.g. 500 to 1,000), the classifier performance closely approximate that using the full feature set, showing that the feature set can be effectively reduced without impairing the performance. Then the three methods are compared in Fig. 2.3b and Fig. 2.3c. For the L-1 method, we train a L-2 regularized SVM using the features selected by the L-1 regularized SVM. The empirical loss on the training set and the regularized loss (empirical loss + regularization), i.e. the value of the objective function, are plot against the number of selected features in Fig. 2.3b. The area under the ROC curve (AUC) for the testing data is shown in Fig. 2.3c. It can be observed that the performance of FFS is very close to RFE, both outperforms the L-1 method. FFS is more favorable than RFE in case that the target number of selected features is much smaller than the total number of features, since RFE starts with high dimensional feature space, and training with high dimensional feature space is computational expensive. Furthermore, FFS provides a fine grained control of the number of selected features; we may add one single feature in each iteration. By contrast, if we remove one feature each time in RFE, the computational cost will be too high. Similar conclusions can be drawn for the L-2 regularized logistic regression, for which the results are shown in Fig. 2.3d, Fig. 2.3e, and Fig. 2.3f.

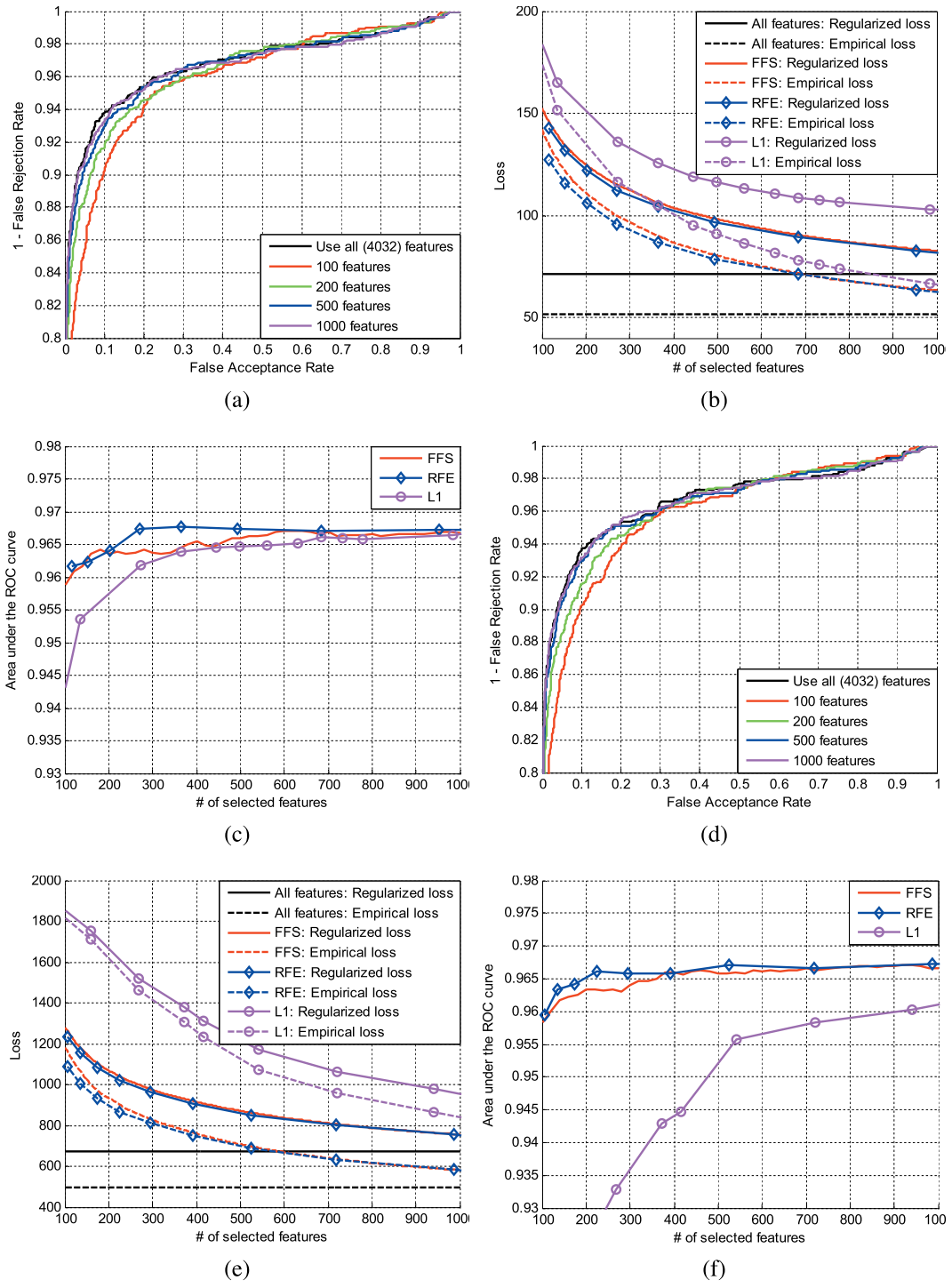


Figure 2.3: (a) (d) ROC curves, (b) (e) empirical loss, and (c) (f) area under the ROC curves for SVM (a) (b) (c) and logistic regression (d) (e) (f), using various feature selection methods and selecting various number of features.

2.5 Problems other than classification

Besides the binary classification problem, some other learning problems are also useful in visual object detection. In this section we discuss two of such problems, i.e. predicting a score for ranking the priority of a number of instances, and structural prediction that predicts more complex labels than the class identity.

2.5.1 Ranking

The ranking problem can be formulated as follows. Let r_a be the ranking priority of an instance with feature $\mathbf{x}_a$, we want to find a function $f(\mathbf{x})$, such that the scores reflect the relative ranking priority between different instances. We expect $f(\mathbf{x}_a) > f(\mathbf{x}_b)$ if $r_a > r_b$, i.e. instance a has higher priority than instance b. The ranking loss function can be defined for a pair of instances:

$$l(a,b) = \begin{cases} 0 & r_a > r_b, f(\mathbf{x}_a) > f(\mathbf{x}_b), \text{i.e. } f(\mathbf{x}_a) - f(\mathbf{x}_b) > 0 \\ 1 & r_a > r_b, f(\mathbf{x}_a) \leq f(\mathbf{x}_b), \text{i.e. } f(\mathbf{x}_a) - f(\mathbf{x}_b) \leq 0 \end{cases} \tag{2.95}$$

Therefore the ranking can be achieved by classifying pairs of instances, which we call the ranking instances. A ranking instance (a,b) is assigned a label equal to $y_{ab} = sign(r_a - r_b)$. If the ranking pair is not interesting, we simply set $y_{ab} = 0$, such that this pair would not be considered in the learning problem. Using a linear prediction function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the feature for a ranking instance can be written as $\mathbf{x}_{ab} = \mathbf{x}_a - \mathbf{x}_b$.

All the previously discussed techniques for solving binary classification problems can be applied to this ranking problem. However, given a training set of M instances, there would be $M \times M$ ranking instances available. The empirical loss is:

$$L = \sum_{i=1}^{M}\sum_{j=1}^{M} l(i,j)$$

This poses difficulty to the learning process since the number of training instances can be huge. How to avoid taking all the instances into consideration would be essential for an efficient learning algorithm. For ranking SVM, both the n-slack cutting plane algorithm and the 1-slack cutting plane algorithm in section 2.3.4 can be applied. However, when the number of training pairs is huge, usually the number of support vectors of ranking pairs is also huge. For example, we find that the number of hard ranking pairs

is usually on the order of millions for the INRIA classification test with 2,474 positives and 10,000 negatives evaluated in section 2.4.3, which cannot be fit into memory. Therefore the 1-slack cutting plane algorithm is better off for the ranking problems. The unconstrained ranking-SVM problem can be written as:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\tfrac{1}{M_P}\sum_{(a,b)\in P}\max\left(0, 1-\mathbf{w}^T\mathbf{x}_{ab}\right), \tag{2.96}$$

where P is the set of ranking pairs (a, b) such that a has higher priority than b, i.e. $y_{ab} = 1$, and M_P is the size of P. The 1-slack formulation is:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi, \\ \text{s.t.} \quad & \forall \boldsymbol{\pi} \in \{0,1\}^{M_P}, \mathbf{w}^T\tfrac{1}{M_P}\sum_{(i,j)\in P}\pi_{ab}\mathbf{x}_{ij} \geq \tfrac{1}{M_P}\sum_{(a,b)\in P}\pi_{ab} - \xi, \\ & \xi \geq 0. \end{aligned} \tag{2.97}$$

However, to add the cutting planes, the indicator vector $\boldsymbol{\pi}$ should be calculated for a particular $\mathbf{w}_t$ and for all pairs (a, b):

$$\pi_{t,ab} = \begin{cases} 1 & \mathbf{w}^T\mathbf{x}_{ab} \leq 1 \\ 0 & \mathbf{w}^T\mathbf{x}_{ab} > 1 \end{cases} \tag{2.98}$$

Naively evaluate this for all $(a, b) \in P$ would involve $O(M_p)$, i.e. $O(M^2)$ complexity. To avoid this cost, a method is proposed in [106]. An observation is that the constraints in (2.98) can be written as:

$$\mathbf{w}^T\frac{1}{M_P}\sum_{i=1}^{M}\left(c_i^+ - c_i^-\right)\mathbf{x}_i \geq \frac{1}{2M_P}\sum_{i=1}^{M}\left(c_i^+ + c_i^-\right) - \xi, \tag{2.99}$$

where c_i^+ is the number of times a training instance $\mathbf{x}_i$ appears as the high priority term for a in-margin pair in P, and c_i^- is the number of times $\mathbf{x}_i$ appears as the low priority instance for a in-margin pair in P, i.e.:

$$c_i^+ = \left|\left\{(i,b) : (i,b) \in P, \mathbf{w}^T\mathbf{x}_{ib} \leq 1\right\}\right| \tag{2.100}$$

$$c_i^- = \left|\left\{(a,i) : (a,i) \in P, \mathbf{w}^T\mathbf{x}_{ai} \leq 1\right\}\right| \tag{2.101}$$

The single-pass Algorithm 1 can be used to decide c_i^+ and c_i^- for all i in $O(M)$. The algorithm acts like a pair of pointers moving along the line of sorted $\mathbf{w}^T\mathbf{x}_i$. The pointer i indicates the current instance under consideration. The pointer j moves until the

difference between the score of i and j exceeds 1, such that for any $k \geq j$, $\mathbf{w}^T\mathbf{x}_{ik} > 1$, and $\pi_{ik} = 0$. As j moves, the number of misclassified pairs in which i is the first term is cumulated in counter b. As i moves, the number of correctly classified pairs with j as the second term is cumulated in counter a.

Algorithm 1 Find the cutting plane for ranking SVM.

$\mathbf{S}$ is the vector of $\{s_i = \mathbf{w}^T\mathbf{x}_i\}$ sorted in descending order.
$\mathbf{I}$ is the indexes of the elements in $\mathbf{S}$ in the original set of training instances.
r_{I_i} is ranking priority of the instance with index i in $\mathbf{S}$.
R is the set of ranking priorities, e.g. $R = \{1, 2\}$ for binary classification.
M_r is the number of instances with priority r.
Initialize $\mathbf{c}^+ = 0$, $\mathbf{c}^- = 0$.
for $r = 2, 3, \ldots$ **do**
  $a = 0, b = 0, i = 1, j = 1$.
  while $i \leq M$ **do**
    if $r_{I_i} = r$ **then**
      while $\mathbf{S}_i - \mathbf{S}_j \leq 1$ AND $j \leq M$ **do**
        if $r_{I_i} > r_{I_j}$ **then**
          $b++, c^-_{I_j} = c^-_{I_j} + M_r - a$.
        end if
        $j++$
      end while
      $a++, c^+_{I_i} = c^+_{I_i} + b$
    end if
  end while
end for
return $\mathbf{c}^+$ and $\mathbf{c}^-$.

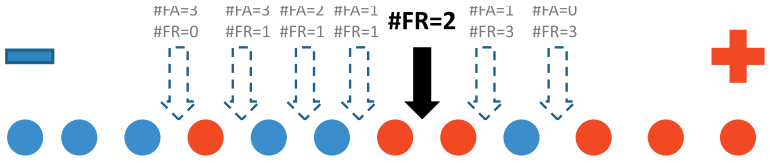


Figure 2.4: The number of swapped pairs is related to the area under the ROC curve.

The ranking loss is an upper bound of the switched pairs, and the number of switched pairs is proportional to the area under the ROC curve:

$$\begin{aligned} ROCArea &= \int (1 - FRR)\, dFAR = 1 - \frac{1}{n_{ROC} M_+} \sum\nolimits_i \#FR_i \\ &= 1 - \frac{1}{n_{ROC} M_+} \#SwappedPair, \end{aligned} \tag{2.102}$$

where n_{ROC} is the number of unique points on the ROC curve, the summation is over the unique ROC points, M_+ is the number of positive instances, and $\#FR_i$ is the number of false rejections at each ROC point. An illustration of the relation between the ROCArea and the swapped pairs is given in Fig. 2.4.

2.5.2 Structural prediction

In the structural prediction problem, we want to predict the label $\mathbf{y}$ of an instance with feature $\mathbf{x}$, but the label is more complex than the binary class identity. For example, predict the class identity from > 2 classes, estimate the human body pose represented by the body joint positions, etc. We want to learn a scoring function $f(\mathbf{y}, \mathbf{x})$, such that the correct label $\mathbf{y}$ scores higher than any other label $\mathbf{y}'$. Then the label of a test instance can be inferred by maximizing the scoring function over the domain of $\mathbf{y}$. Different from the classification problem and the ranking problem, the function f now takes a label-feature pair $(\mathbf{y}, \mathbf{x})$ as the input, and generates an internal representation $\Phi(\mathbf{y}, \mathbf{x})$ for the pair, which is called a joint feature mapping. For example, in the human pose estimation problem, the image features are extracted according to the location of each body part. Furthermore, the label $\mathbf{y}$ may contribute to the joint feature mapping $\Phi(\mathbf{y}, \mathbf{x})$ by itself. An analogy of the scoring function using the joint feature mapping is the joint probability $P(\mathbf{x}, \mathbf{y}) \propto P(\mathbf{x}|\mathbf{y}) P(\mathbf{y})$, where $\mathbf{y}$ contributes to the joint probability in both the likelihood term $P(\mathbf{x}|\mathbf{y})$ and the prior term $P(\mathbf{y})$. For example, prior knowledge can be crucial in estimating the body pose.

The structural prediction problem is studied in [48]. The structural prediction problem can be cast as a ranking problem, such that we would like the correct labeling $(\mathbf{y}, \mathbf{x})$ ranks higher than any other labeling $(\mathbf{y}', \mathbf{x})$, i.e. $f(\mathbf{y}, \mathbf{x}) > f(\mathbf{y}', \mathbf{x})$, $\forall \mathbf{y}' \neq \mathbf{y}$. Furthermore, it is desirable that the difference between $f(\mathbf{y}, \mathbf{x})$ and $f(\mathbf{y}', \mathbf{x})$ reflects how different $\mathbf{y}$ and $\mathbf{y}'$ are, or the cost of confusing $\mathbf{y}'$ for $\mathbf{y}$ is proportional to how different they are. Two approaches can be employed to take this into consideration. First, we may require $f(\mathbf{y}, \mathbf{x})$ to be larger than $f(\mathbf{y}', \mathbf{x})$ by a margin proportional to $\Delta(\mathbf{y}, \mathbf{y}')$, which measures the difference between $\mathbf{y}$ and $\mathbf{y}'$. If the margin fails to be satisfied, an error occurs. This results in the following margin-scaling loss function:

$$l(\mathbf{x}, \mathbf{y}, \mathbf{y}') = \begin{cases} 0 & f(\mathbf{y}, \mathbf{x}) > f(\mathbf{y}', \mathbf{x}) + \Delta(\mathbf{y}, \mathbf{y}') \\ 1 & f(\mathbf{y}, \mathbf{x}) \leq f(\mathbf{y}', \mathbf{x}) + \Delta(\mathbf{y}, \mathbf{y}') \end{cases} \tag{2.103}$$

Second, we may penalize an error, i.e. $f(\mathbf{y}, \mathbf{x}) \leq f(\mathbf{y}', \mathbf{x})$ by a cost proportional to

$\Delta(\mathbf{y}, \mathbf{y}')$, resulting in the following loss-scaling loss function:

$$l(\mathbf{x}, \mathbf{y}, \mathbf{y}') = \begin{cases} 0 & f(\mathbf{y}, \mathbf{x}) > f(\mathbf{y}', \mathbf{x}) \\ \Delta(\mathbf{y}, \mathbf{y}') & f(\mathbf{y}, \mathbf{x}) \leq f(\mathbf{y}', \mathbf{x}) \end{cases} \tag{2.104}$$

In learning the structural prediction model, we are given a training set that the groundtruth label of each instance is known: $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{M}$. For each training instance, there could be $|\mathcal{Y}| - 1$ incorrect labels $\hat{\mathbf{y}}_i'$. Instead of considering all incorrect labels in the loss function, only the one with highest loss is considered. Therefore the empirical loss is:

$$L = \sum_{i=1}^{M} \max_{\mathbf{y'}_i \in \mathcal{Y}} \{l(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}}_i')\}. \tag{2.105}$$

Using margin scaling and hinge loss, the learning problem written in constrained form is:

$$\begin{aligned} \min_{\mathbf{w}, \boldsymbol{\xi}} \quad & \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \xi_i \\ \text{s.t.} \quad & \forall \hat{\mathbf{y}}_i', \mathbf{w}^T(\Phi(\mathbf{y}_i, \mathbf{x}_i) - \Phi(\hat{\mathbf{y}}_i', \mathbf{x}_i)) \geq \Delta(\mathbf{y}_i, \hat{\mathbf{y}}_i') - \xi_i, \ i = 1, ..., M, \\ & \xi_i \geq 0, \ i = 1, ..., M. \end{aligned} \tag{2.106}$$

There are $M \times (|\mathcal{Y}| - 1)$ constraints since $\hat{\mathbf{y}}_i'$ may take any value in $\mathcal{Y}$. However, for each training instance, only the most violated constraint is active. Therefore we only need to consider the most violated constraint for each training instance. A strategy similar to the n-slack cutting plane method in section 2.3.4 can be employed. We iterate between two stages:

- Find the most violated labeling $\hat{\mathbf{y}}_i'$ for each labeled instance $(\mathbf{x}_i, \mathbf{y}_i)$ using the current solution $\mathbf{w}_t$:

$$\hat{\mathbf{y}}_{i,t}' = \arg\max_{\mathbf{y}_i'} \left\{\Delta(\mathbf{y}_i, \hat{\mathbf{y}}_i') - \mathbf{w}_t^T(\Phi(\mathbf{y}_i, \mathbf{x}_i) - \Phi(\hat{\mathbf{y}}_i', \mathbf{x}_i))\right\} \tag{2.107}$$

- Only keep the most violated constraint for each training instance, and solve the optimization problem.

The 1-slack cutting plane method in section 2.3.4 can also be applied to structural learning, as done in [109]. The constrained form (2.106) is equivalent to the following

unconstrained form:

$$\min_{\mathbf{w}} \quad \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{M} \max\left(0, \max_{\hat{\mathbf{y}}'_i} \left\{\Delta\left(\mathbf{y}_i, \hat{\mathbf{y}}'_i\right) - \mathbf{w}^T\left(\Phi\left(\mathbf{y}_i, \mathbf{x}_i\right) - \Phi\left(\hat{\mathbf{y}}'_i, \mathbf{x}_i\right)\right)\right\}\right) \tag{2.108}$$

A lower-bound of the empirical loss in (2.105) can be formulated using the cutting planes:

$$L\left(\mathbf{w}\right) \geq \max\left(0, \max_{t=1,...,T} \left\{C_t\left(\mathbf{w}\right)\right\}\right), \tag{2.109}$$

where $C_t\left(\mathbf{w}\right)$ is the cutting plane of $L\left(\mathbf{w}\right)$, evaluated at $\mathbf{w}_t$:

$$C_t\left(\mathbf{w}\right) = \sum_{i=1}^{M} \pi_{t,i}\left(\Delta\left(\mathbf{y}_i, \hat{\mathbf{y}}'_{i,t}\right) - \mathbf{w}^T\left(\Phi\left(\mathbf{y}_i, \mathbf{x}_i\right) - \Phi\left(\hat{\mathbf{y}}'_{i,t}, \mathbf{x}_i\right)\right)\right), \tag{2.110}$$

and $\hat{\mathbf{y}}'_{i,t}$ is the most incorrect labeling using current $\mathbf{w}_t$ as in (2.107). The indicators $\{\pi_{t,i}\}$ are defined as follows:

$$\pi_{t,i} = \begin{cases} 0, & \mathbf{w}_t^T\left(\Phi\left(\mathbf{y}_i, \mathbf{x}_i\right) - \Phi\left(\hat{\mathbf{y}}'_{i,t}, \mathbf{x}_i\right)\right) \geq \Delta\left(\mathbf{y}_i, \hat{\mathbf{y}}'_{i,t}\right), \\ 1, & \text{otherwise.} \end{cases} \tag{2.111}$$

Then the sub problem solved in each cutting plane iteration is:

$$\min_{\mathbf{w}} \quad \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \max\left(0, \max_{t=1,...,T} \left\{C_t\left(\mathbf{w}\right)\right\}\right). \tag{2.112}$$

Written in the slacked form, the sub problem is:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \xi \\ \text{s.t.} \quad & \sum_{i=1}^{M} \pi_{t,i}\mathbf{w}^T\left(\Phi\left(\mathbf{y}_i, \mathbf{x}_i\right) - \Phi\left(\hat{\mathbf{y}}'_{i,t}, \mathbf{x}_i\right)\right) \geq \sum_{i=1}^{M} \pi_{t,i}\Delta\left(\mathbf{y}_i, \hat{\mathbf{y}}'_{i,t}\right) - \xi, \ t = 1, ..., T, \\ & \xi \geq 0. \end{aligned} \tag{2.113}$$

The dual problem is a QP with only T variables, and thus this problem is much easier to solve than problem (2.106).

2.5.3 Application in sliding window detection

Learning a sliding window detector can be cast as a ranking or structural prediction problem. As a ranking problem, the detector assigns a ranking priority to each window, and a object should have higher priority than all the background windows, while

the ranking order between two objects or two background windows is not important. The training procedure using the ranking problem can be identical to that using the classification problem, by forming ranking pairs between positive instances and negative instances from the negative images. In addition, the ranking can be addressed by forming ranking pairs of a positive instance and a window from the neighborhood of the positive instance, for example, requiring the groundtruth window ranks higher than any window that does not significantly overlap with the groundtruth.

The overlap between the groundtruth object window and an arbitrary other window can be further exploited by treating detection as a structural prediction problem, e.g. [110]. The goal is to predict the location $\mathbf{y}$ of an object inside the image $\mathbf{x}$. The object label $\mathbf{y}$ represents the bounding box of the object inside the image. The joint feature mapping $\Phi(\mathbf{y}, \mathbf{x})$ simply extracts image features for the window $\mathbf{y}$ in the image $\mathbf{x}$. Given a set of object windows, the incorrect labeling for each training instance consists of all windows other than the groundtruth bounding box, but another object window is not counted as a incorrect labeling. The distance function $\Delta(\mathbf{y}, \mathbf{y}')$ measures how much the incorrect bounding box $\mathbf{y}'$ differs from the groundtruth bounding box $\mathbf{y}$ according to their overlap ratio. Therefore the overlap ratio can be better exploited, and improvement in detection performance can be expected. Furthermore, large dataset can be exploited using the cutting plane training for ranking SVM and structural SVM.

In practice, we find that the detector learned by ranking SVM or structural SVM performs similarly to that learned by classification SVM, all trained for the pedestrian detection problem using the INRIA dataset. All the dozens of millions of windows in the negative training images are exploited using the n-slack cutting plane method for hard negative mining. This implies that given sufficient training data, the normal direction of the max-margin separating hyperplane is very close to the optimal projection direction for ranking or structural prediction. However, as reported in other works, e.g. [110], better detectors can be obtained by ranking or structural prediction when the amount of training data is small.

Chapter 3 The Boosting algorithm

The boosting algorithm is studied in this chapter. First we introduce the Boosting algorithm as a convex optimization problem that finds the optimal linear combination coefficients of weak learners in Section 3.1, and as a functional optimization problem in Section 3.2. Then two improvements for the Boosting algorithm are proposed. The continuous-valued weak learners derived from treating Boosting as functional optimization is plugged into the totally-corrective Boosting due to the convex optimization formulation for better performance in Section 3.3, and the training process is accelerated by a feature subset selection method based on the partial least square (PLS) regression in Section 3.4.

3.1 Boosting as convex optimization

Boosting is a meta-algorithm for supervised learning. It combines the outputs of a number of weak learners $\{f_i(\mathbf{x})\}$ to obtain a single strong learner $F(\mathbf{x})$. The weak learners are required to perform just slightly better than random guess, while the strong learner can be very accurate. The strong learner is the linear combination of the weak learners:

$$F(\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(\mathbf{x}) \tag{3.1}$$

As introduced in section 2.1, the generalization performance of Boosting algorithms is only related to the VC dimension of the weak learners and the distribution of margins of the training instances:

Theorem. *Let $\mathcal{H}$ be the set of base classifiers (weak classifiers), and with VC dimension d. Let $F(\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(\mathbf{x})$ be the strong classifier obtained by linear combination of weak classifiers from $\mathcal{H}$. M is the training set size. Then with probability $1-\eta$:*

$$err_D \leq P_S\left(y\frac{F(x)}{\sum_{t=1}^{T}|w_t|} \leq \Delta\right) + O\left(\frac{1}{\sqrt{M}}\sqrt{\frac{d\log^2(M/d)}{\Delta^2} + \log\frac{1}{\eta}}\right)$$

For a given value of the margin Δ, $P_S\left(y\frac{F(x)}{\sum_{t=1}^{T}|w_t|} \leq \Delta\right)$ can be approximated by the empirical loss, e.g. the hinge loss, the logistic loss, and the exponential loss. Then the Boosting learning problem can be formulated as:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \sum_{i=1}^{M} l\left(y_i, F\left(\mathbf{x}_i\right); \Delta\right) \\ \text{s.t.} \quad & F\left(\mathbf{x}_i\right) = \sum_{t=1}^{T} w_t f_t\left(\mathbf{x}_i\right), \\ & \sum_{t=1}^{T} |w_t| = 1. \end{aligned} \tag{3.2}$$

Notice that the loss function l is parameterized by the target margin Δ. Due to the linear dependency between Δ and $\sum_{t=1}^{T}|w_t|$, we may fix Δ, e.g. $\Delta = 1$, and varying the L-1 norm of $\mathbf{w}$, and the problem becomes:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \sum_{i=1}^{M} l\left(y_i, F\left(\mathbf{x}_i\right), 1\right) \\ \text{s.t.} \quad & F\left(\mathbf{x}_i\right) = \sum_{t=1}^{T} w_t f_t\left(\mathbf{x}_i\right), \\ & \sum_{t=1}^{T} |w_t| = C. \end{aligned} \tag{3.3}$$

Written in the Lagrangian form, the problem is:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \lambda \sum_{t=1}^{T} |w_t| + \sum_{i=1}^{M} l\left(y_i, F\left(\mathbf{x}_i\right)\right) \\ \text{s.t.} \quad & F\left(\mathbf{x}_i\right) = \sum_{t=1}^{T} w_t f_t\left(\mathbf{x}_i\right), \end{aligned} \tag{3.4}$$

Given the set of base learners $\{f_1\left(\mathbf{x}\right), ..., f_N\left(\mathbf{x}\right)\}$, the combination weights can be obtained by solving the optimization problem (3.4). Furthermore, the forward feature selection technique discussed in section 2.4 can be applied such that the weak learners are sequentially included into the strong learner. A new weak learner is selected into the strong learner according to the following criterion:

$$\arg\max_{j} \left| \sum_{i=1}^{M} \alpha_i f_j\left(\mathbf{x}_i\right) \right|, \tag{3.5}$$

where $\{\alpha_i\}$ are the dual variables for the training instances, and can be interpreted as instance weights. For differentiable losses, e.g. exponential loss and logistic loss, $\{\alpha_i\}$

can be obtained in closed form from the current strong classifier score:

$$\begin{aligned}
\text{Exponential loss} \qquad & l\left(y_i, F\left(\mathbf{x}_i\right)\right) = \exp\left(-y_i F\left(\mathbf{x}_i\right)\right) \\
& \alpha_i = -\frac{\partial l\left(y_i, F\left(\mathbf{x}_i\right)\right)}{\partial F\left(\mathbf{x}_i\right)} = y_i \exp\left(-y_i F\left(\mathbf{x}_i\right)\right) \\
\text{Logistic loss} \qquad & l\left(y_i, F\left(\mathbf{x}_i\right)\right) = \log\left(1 + \exp\left(-y_i F\left(\mathbf{x}_i\right)\right)\right) \\
& \alpha_i = -\frac{\partial l\left(y_i, F\left(\mathbf{x}_i\right)\right)}{\partial F\left(\mathbf{x}_i\right)} = \frac{y_i \exp\left(-y_i F\left(\mathbf{x}_i\right)\right)}{1 + \exp\left(-y_i F\left(\mathbf{x}_i\right)\right)}
\end{aligned}$$

In addition, the sign of the weak learners can be inversed such that $w_j \geq 0$ for all weak learners, and the weak learner selection criterion becomes:

$$\arg\max_j \sum\nolimits_{i=1}^{M} \alpha_i f_j\left(x_i\right). \tag{3.6}$$

Then the learning problem can be modified to:

$$\begin{aligned}
\min_{\mathbf{w}} \quad & \lambda \sum_{t=1}^{T} w_t + \sum_{i=1}^{M} L\left(y_i, F\left(\mathbf{x}_i\right)\right) \\
\text{s.t.} \quad & F\left(\mathbf{x}_i\right) = \sum_{t=1}^{T} w_t f_t\left(\mathbf{x}_i\right), \\
& w_t \geq 0.
\end{aligned} \tag{3.7}$$

Boosting algorithms that alternate between selecting weak learners and solving for $\mathbf{w}$ are known as *totally-corrective Boosting* algorithms as in [68], since all weak learners' weights are updated by solving problem (3.7). In practice we find that L-2 regularization works equally well as the L-1 regularization in terms of generalization performance. The new weak classifier selection criterion is identical for L-1 and L-2 regularization, but the L-2 regularized problem (3.8) is an unconstrained optimization problem with smooth and convex objective, and can be easily solved by Newton's method, which is much easier than (3.7). The only benefit of (3.7) seems to be in the inherent sparsity induced by the L-1 regularization/constraint. However, the Boosting has explicitly performed forward feature selection, making the additional sparsity due to the L-1 technique unnecessary.

$$\min_{\mathbf{w}} \quad \lambda \sum_{t=1}^{T} w_t^2 + \sum_{i=1}^{M} L\left(y_i, \sum_{t=1}^{T} w_t f_t\left(\mathbf{x}_i\right)\right) \tag{3.8}$$

Furthermore, the L-1 regularized formulation will stop adding new weak classifiers if there is no violated dual constraint, i.e. $\max_j \sum_{i=1}^{M} \alpha_i f_j\left(x_i\right) \leq \lambda$, which does not

happen for the L-2 regularized form.

3.2 Boosting as functional optimization

The Boosting algorithms are also interpreted from the functional optimization perspective. Instead of solving for the combination coefficients of a number of weak learners, we want to minimize the expected loss on the distribution D, with respect to the unknown function $F(\mathbf{x})$:

$$\min_{F} \quad E_{(\mathbf{x},y)\sim D}\left(L\left(y, F\left(\mathbf{x}\right)\right)\right). \tag{3.9}$$

Again, the expected loss can be approximated by the empirical loss:

$$\min_{F} \quad \sum_{i=1}^{M} l\left(y_i, F\left(\mathbf{x}_i\right)\right). \tag{3.10}$$

Starting with $F(\mathbf{x}) = 0$, we proceed by modifying the function in an additive manner:

$$F\left(\mathbf{x}\right) \leftarrow F\left(\mathbf{x}\right) + wf\left(\mathbf{x}\right). \tag{3.11}$$

Therefore in each step we solve the following problem:

$$\min_{f,w} \quad E_{(\mathbf{x},y)\sim D}\left(L\left(y, F\left(\mathbf{x}\right) + wf\left(\mathbf{x}\right)\right)\right). \tag{3.12}$$

In this section we introduce three popular Boosting variants using the exponential loss $\exp\left(-yF\left(\mathbf{x}\right)\right)$. All the three algorithms solve the function optimization problem (3.9), using different methods.

3.2.1 Discrete AdaBoost

In the following we denote $\mathbf{F}$ the vector of strong learner scores for all training instances, $\mathbf{f}$ the vector of weak learner scores for all training instances, and L the sum of exponential loss on the training set. We take the 2nd order approximation of the loss:

$$L\left(\mathbf{F} + w\mathbf{f}\right) \approx L\left(\mathbf{F}\right) + w\mathbf{f}^T G + \frac{1}{2} w^2 \mathbf{f}^T H \mathbf{f}, \tag{3.13}$$

where the gradient G and Hessian H are computed as follows:

$$\begin{aligned} G_i &= \left.\frac{\partial L}{\partial f_i}\right|_{f_i=0} = -y_i \exp\left(-y_i F_i\right), \\ H_{ii} &= \left.\frac{\partial^2 L}{\partial {f_i}^2}\right|_{f_i=0} = \exp\left(-y_i F_i\right), \\ H_{ij} &= 0, i \neq j. \end{aligned} \tag{3.14}$$

If we limit our choice of weak learners to binary classifiers, i.e. $f_i \in \{+1, -1\}$, then $\frac{1}{2} w^2 \mathbf{f}^T H \mathbf{f}$ is constant for all $\mathbf{f}$. Therefore the approximation (3.13) is minimized by:

$$\begin{aligned} \mathbf{f}^* &= \arg\min_{\mathbf{f}} \mathbf{f}^T G \\ &= \arg\min_{\mathbf{f}} \sum_{i=1}^{M} -y_i \exp\left(-y_i F_i\right) f_i \\ &= \arg\min_{\mathbf{f}} \sum_{i=1}^{M} -\alpha_i f_i \end{aligned} \tag{3.15}$$

This problem is interpreted as finding the weak classifier that minimizes a weighted loss, with the weight for each training instance being $\omega_i = \exp\left(-y_i F_i\right)$. Since $\alpha_i = y_i \exp\left(-y_i F_i\right)$ is the negative gradient of the loss function, therefore the problem can also be interpreted as finding the weak classifier that best approximates the negative gradient direction.

Then the weight w of the new weak classifier can be found in closed form, by minimizing $L\left(\mathbf{F} + w\mathbf{f}\right)$ with respect to w:

$$w = \frac{1}{2} \log \frac{Z - err}{err} = \frac{1}{2} \log \frac{1 - err^*}{err^*}, \tag{3.16}$$

where Z is the sum of weights and err is the weighted error:

$$\begin{aligned} Z &= \sum_{i=1}^{M} \exp\left(-y_i F\left(\mathbf{x}_i\right)\right), \\ err &= \sum_{y_i \neq f(\mathbf{x}_i)} \exp\left(-y_i F\left(\mathbf{x}_i\right)\right), \\ err^* &= \frac{err}{Z}. \end{aligned} \tag{3.17}$$

We can expect the problem becomes more and more difficult, and err^* approaches 0.5, thus the later weak classifiers will have lower weights, as illustrated in Fig. 3.1.

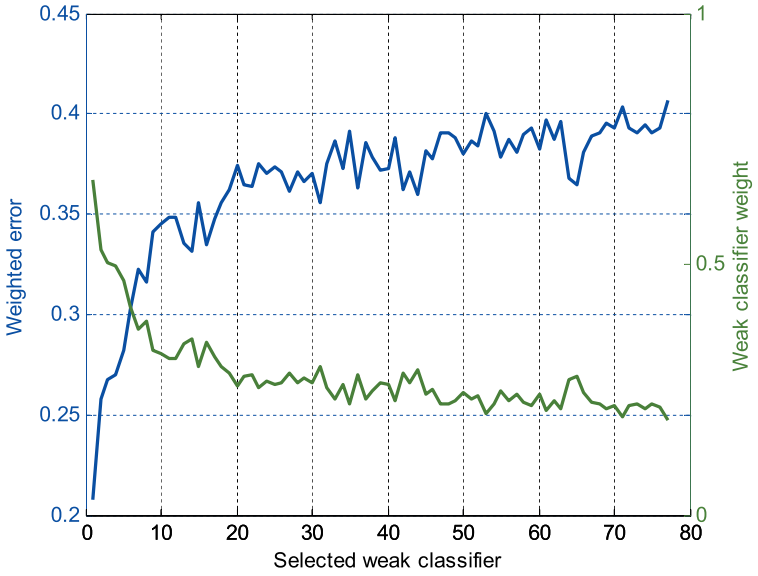


Figure 3.1: As the strong classifier becomes stronger, the weighted data becomes more difficult, the weighted error of the best weak classifier increases, and the weak classifier will have lower weight.

Therefore the sum of weights $\sum_{t=1}^{T} w_t$ can be constrained by selecting a finite number of weak learners.

3.2.2 Real AdaBoost

Instead of using binary classifiers as weak learners, the weak learner set $\mathcal{H}$ may include continuous-valued functions. Therefore we may set the combination weight w to 1, and look for a function f that minimizes the expected loss functional:

$$\min_{f} \quad L = E_D \left(L \left(y, F \left(\mathbf{x} \right) + f \left(\mathbf{x} \right) \right) \right) \tag{3.18}$$

Taking the derivative of the loss functional, and set it to 0:

$$\begin{aligned} \frac{\partial L}{\partial f \left(\mathbf{x} \right)} &= E_D \left(e^{-yF(\mathbf{x})} \left(-y \right) e^{-yf(\mathbf{x})} \right) \\ &= P_D \left(y = -1, \mathbf{x} \right) e^{F(\mathbf{x})} e^{f(\mathbf{x})} - P_D \left(y = 1, \mathbf{x} \right) e^{-F(\mathbf{x})} e^{-f(\mathbf{x})} \\ &= P_{\boldsymbol{\omega}} \left(y = -1, \mathbf{x} \right) e^{f(\mathbf{x})} - P_{\boldsymbol{\omega}} \left(y = 1, \mathbf{x} \right) e^{-f(\mathbf{x})} \\ &= 0 \end{aligned} \tag{3.19}$$

where $\boldsymbol{\omega}$ specifies a weighted distribution, with instance weights $\omega_i = e^{-y_i F(\mathbf{x}_i)}$. Therefore we can obtain the minimizer of (3.18) in closed form:

$$f^* (\mathbf{x}) = \frac{1}{2} \log \frac{P_D (y = 1, \mathbf{x}) e^{-F(\mathbf{x})}}{P_D (y = -1, \mathbf{x}) e^{F(\mathbf{x})}} = \frac{1}{2} \log \frac{P_{\boldsymbol{\omega}} (y = 1|\mathbf{x})}{P_{\boldsymbol{\omega}} (y = -1|\mathbf{x})} \tag{3.20}$$

Then the new weak learner $f (\mathbf{x})$ can be learned as a regression function, to approximate the following value on the training set:

$$f (\mathbf{x}) \approx \frac{1}{2} \log \frac{P_{\omega} (y = 1|\mathbf{x})}{P_{\omega} (y = -1|\mathbf{x})}. \tag{3.21}$$

Then among all the learned regression functions, we select the one that mostly improves the empirical loss:

$$\begin{aligned} &\min_f \sum_{i=1}^{M} \exp \left(-y_i \left(F (\mathbf{x}_i) + f (\mathbf{x}_i)\right)\right), \\ &\text{i.e. } \min_f \sum_{i=1}^{M} \omega_i \exp \left(-y_i \left(f (\mathbf{x}_i)\right)\right) \end{aligned} \tag{3.22}$$

3.2.3 Gentle AdaBoost

Similar to the Real AdaBoost, the Gentle AdaBoost uses continuous value weak learners. The difference is, instead of directly minimizing the loss, the Gentle AdaBoost uses the Newton's method. The gradient and Hessian of the loss functional is:

$$\begin{aligned} G (\mathbf{x}) &= \frac{\partial L (F (\mathbf{x}))}{\partial F (\mathbf{x})} \\ &= E_D \left(-y e^{-yF(\mathbf{x})}\right) = P_{\boldsymbol{\omega}} (y = -1, \mathbf{x}) - P_{\boldsymbol{\omega}} (y = 1, \mathbf{x}) \qquad (3.23) \\ H (\mathbf{x}) &= \frac{\partial^2 L (F (\mathbf{x}))}{\partial F (\mathbf{x})^2} = E_D \left(e^{-yF(\mathbf{x})}\right) = P_{\boldsymbol{\omega}} (\mathbf{x}) \qquad (3.24) \end{aligned}$$

Therefore the Newton step in the functional space is:

$$\begin{aligned} -H^{-1} G &= \frac{P_{\omega} (y = 1, \mathbf{x}) - P_{\omega} (y = -1, \mathbf{x})}{P_{\omega} (\mathbf{x})} \\ &= P_{\omega} (y = 1|\mathbf{x}) - P_{\omega} (y = -1|\mathbf{x}) \end{aligned} \tag{3.25}$$

The new weak learner is a regression function fitted to the following target value:

$$P_{\omega} (y = 1|\mathbf{x}) - P_{\omega} (y = -1|\mathbf{x}) . \tag{3.26}$$

Similar to the Real AdaBoost, the regressor that minimizes the empirical loss is selected. Compared to the Real AdaBoost, the Gentle AdaBoost is more numerically stable since the weak classifiers' output is limited to $[-1, +1]$, and hence the running weight of the training instances $\omega_i = e^{-y_i F(\mathbf{x}_i)}$ does not take extremely large or small values.

The Gentle AdaBoost can also be directly derived from the 2nd order approximation (3.13) that minimizes the empirical loss rather than the loss functional. But the difference is, now we consider $\mathbf{f}$ to be the output of any continuous-valued function, and the weight of the new weak classifier can be fixed to 1:

$$L(\mathbf{F} + \mathbf{f}) \approx L(\mathbf{F}) + \mathbf{f}^T \mathbf{G} + \frac{1}{2} \mathbf{f}^T H \mathbf{f} \tag{3.27}$$

The minimizer is:

$$\begin{aligned} \mathbf{f}^* &= \arg\min_{\mathbf{f}} \mathbf{f}^T \mathbf{G} + \frac{1}{2} \mathbf{f}^T H \mathbf{f} \\ &= \arg\min_{\mathbf{f}} \left(\mathbf{f} + H^{-1}\mathbf{G}\right)^T H \left(\mathbf{f} + H^{-1}\mathbf{G}\right). \end{aligned} \tag{3.28}$$

Plugging in the Hessian and gradient given in (3.14), we can obtain $H^{-1}\mathbf{G} = -\mathbf{y}$. Therefore training the weak classifier essentially involves solving a weighted least square problem using the feature vector $\mathbf{x}_i$ as input, and the label y_i as the target variable, and using $\exp(-y_i F(\mathbf{x}_i))$ as the instance weights.

3.2.4 Experimental evaluation

We perform a simple experiment to evalaute the performance of Discrete AdaBoost, Real AdaBoost, and Gentle AdaBoost. For the weak learners, we use stumps performing on individual features. A stump is a function of the following form:

$$f(x) = \begin{cases} v_1 & x \geq th \\ v_2 & x < th \end{cases} \tag{3.29}$$

For each feature, an optimal threshold th is selected. For Discrete AdaBoost, the decision stump assigns $v_1 = 1$ and $v_2 = -1$, or vice versa. For Real AdaBoost and Gentle AdaBoost, v_1 and v_2 are selected according to (3.21) or (3.26). Then the weak classifier that minimizes the exponential loss is selected. In addition, to avoid numerical problems in computing the weights $\omega_i = \exp(-y_i F(\mathbf{x}_i))$, the strong classifier score $F(\mathbf{x}_i)$ is truncated to be within $[-10, +10]$ when computing the instance weights.

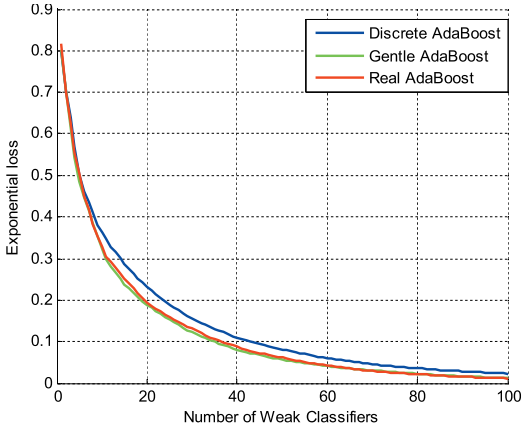


Figure 3.2: Comparison of the convergence speed of Discrete AdaBoost, Real AdaBoost, and Gentle AdaBoost.

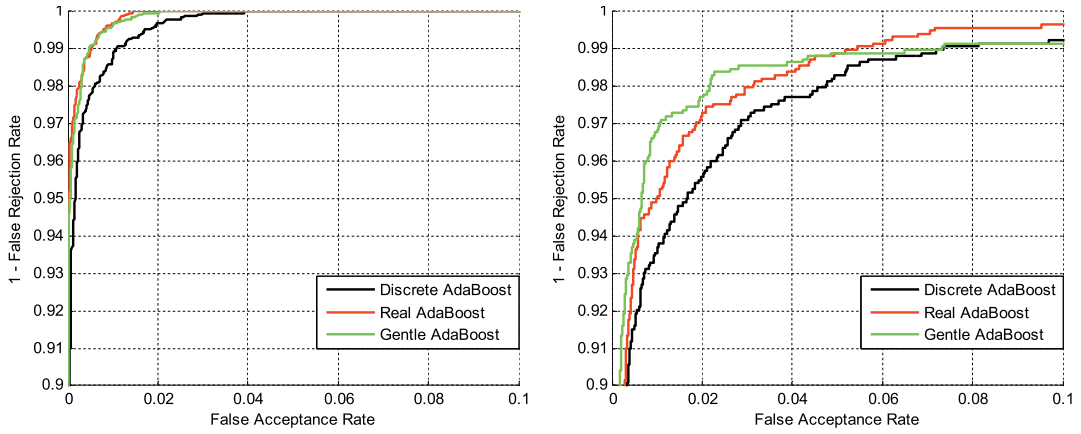


Figure 3.3: Comparison of the ROC curves. (a) ROC curves on the training set. (b) ROC curves on the testing set.

The experiment is performed using the INRIA pedestrian dataset. The training set consists of 2,474 pedestrian images and 10,000 background windows randomly taken from the negative images, and the testing set consists of 1,178 pedestrians and another 10,000 background windows. HOG features are extracted for 619 rectangular blocks, with size 16×16, 32×32, and 64×64, resulting in a feature vector of length 19,808. 100 weak classifiers are selected for each Boosting algorithm. Since all three methods optimizes for the exponential loss functional, we show the decrease of exponential loss as more weak classifiers are added in Fig. 3.2. It can be observed that Real AdaBoost and Gentle AdaBoost has faster convergence rate than Discrete AdaBoost, which is essentially due the extra degree of freedom in the weak classifiers, and the use of sec-

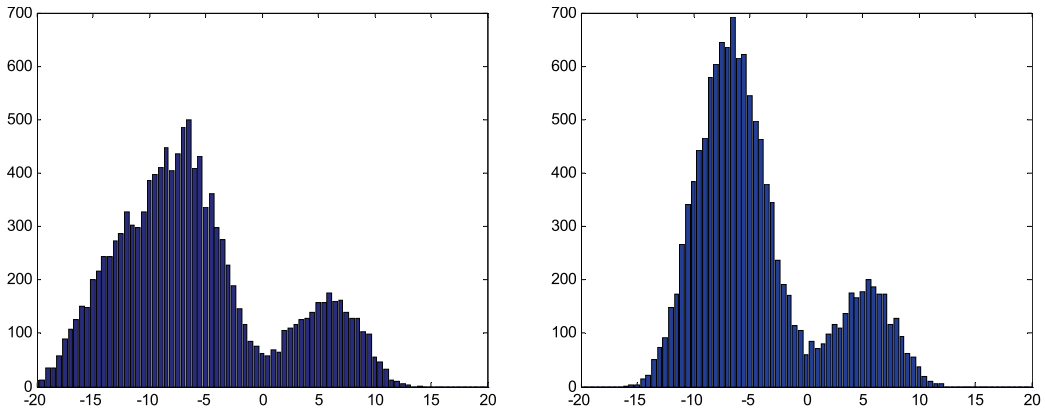


Figure 3.4: (a) Distribution of strong classifier scores for Real AdaBoost. (b) Distribution of strong classifier scores for Gentle AdaBoost.

ond order information in finding the solution. To see the actual performance of each method, the ROC curves on the training set and testing set using 50 weak classifiers are displayed in Fig. 3.3. The Gentle AdaBoost is the best performing method, while the Real AdaBoost is slightly inferior. This is probably due to the Gentle AdaBoost constrains the weak learner output in $[-1, +1]$, while the weak learner output of Real AdaBoost is unconstrained. The distributions of the strong classifier scores on the training set are shown in Fig. 3.4. We can see that the Gentle AdaBoost strong classifier scores span a narrower range than Real AdaBoost, such that the exponential loss better approximates the binary 0-1 loss.

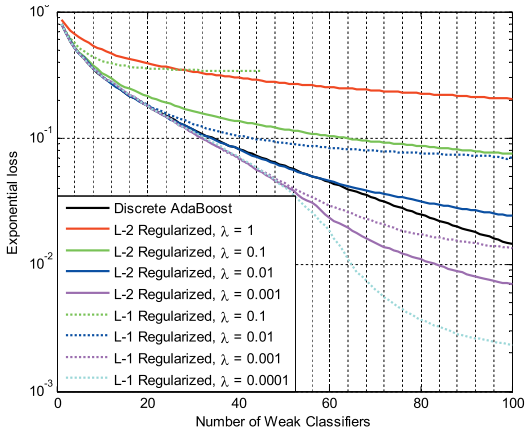


Figure 3.5: Comparison of the exponential loss of L-1 and L-2 regularized totally-corrective AdaBoost.

We also compare the Discrete AdaBoost and the totally-corrective AdaBoost, both

using decision stump weak classifiers. The L-1 regularized form (3.7) and the L-2 regularized form (3.8) are studied. For L-2 regularization, we show the results for $\lambda = 1$, 0.1, 0.01, and 0.001. For L-1 regularization, we show results for $\lambda = 0.01$, 0.001, and 0.0001. For L-1 regularization and $\lambda = 0.1$, the totally-corrective AdaBoost stops after selecting 48 weak classifiers, since no more weak classifier violates the dual constraint in the forward feature selection scheme. The exponential loss are shown in Fig. 3.5. It can be observed that for small λ, the objective function is dominated by the empirical loss, and therefore the totally-corrective methods results in much lower exponential loss and converges faster than the Discrete AdaBoost. For large λ, the regularization term dominates the objective function. The ROC curve on the testing set is shown in Fig. 3.6. The totally-corrective Boosting shows better performance for properly selected λ. Furthermore, using very small λ, i.e. radically minimizing the exponential loss, actually impairs the performance, indicating the importance of regularization. It also turns out that the gain due to totally-corrective Boosting diminishes with a large number of weak classifiers selected (e.g. 100).

3.3 Continuous-valued weak learners for totally-corrective Boosting

3.3.1 Embedding continuous-valued weak learners in totally-corrective AdaBoost

As described in section 3.1, the Boosting algorithm seeks the optimal linear combination coefficients $\mathbf{w}$ to optimize problem (3.7), and the forward feature selection scheme in section 2.3 can be used for sequentially selecting weak learners. The criterion for selecting the weak learner is (3.6), and no more weak learners will be selected if $\max_j \sum_{i=1}^{M} \alpha_i f_j(\mathbf{x}_i) \leq \lambda$, i.e. no dual constraint is violated for problem (3.7). Furthermore, if the loss L is differentiable, the per-instance dual variable α_i is:

$$\alpha_i = -\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}. \tag{3.30}$$

Assume the weak learners have been obtained beforehand, solving the problem (3.6) is referred to as the totally-corrective Boosting, since all the weak learners' weights are adjusted. By comparison, the stage-wise additive method discussed in section 3.2

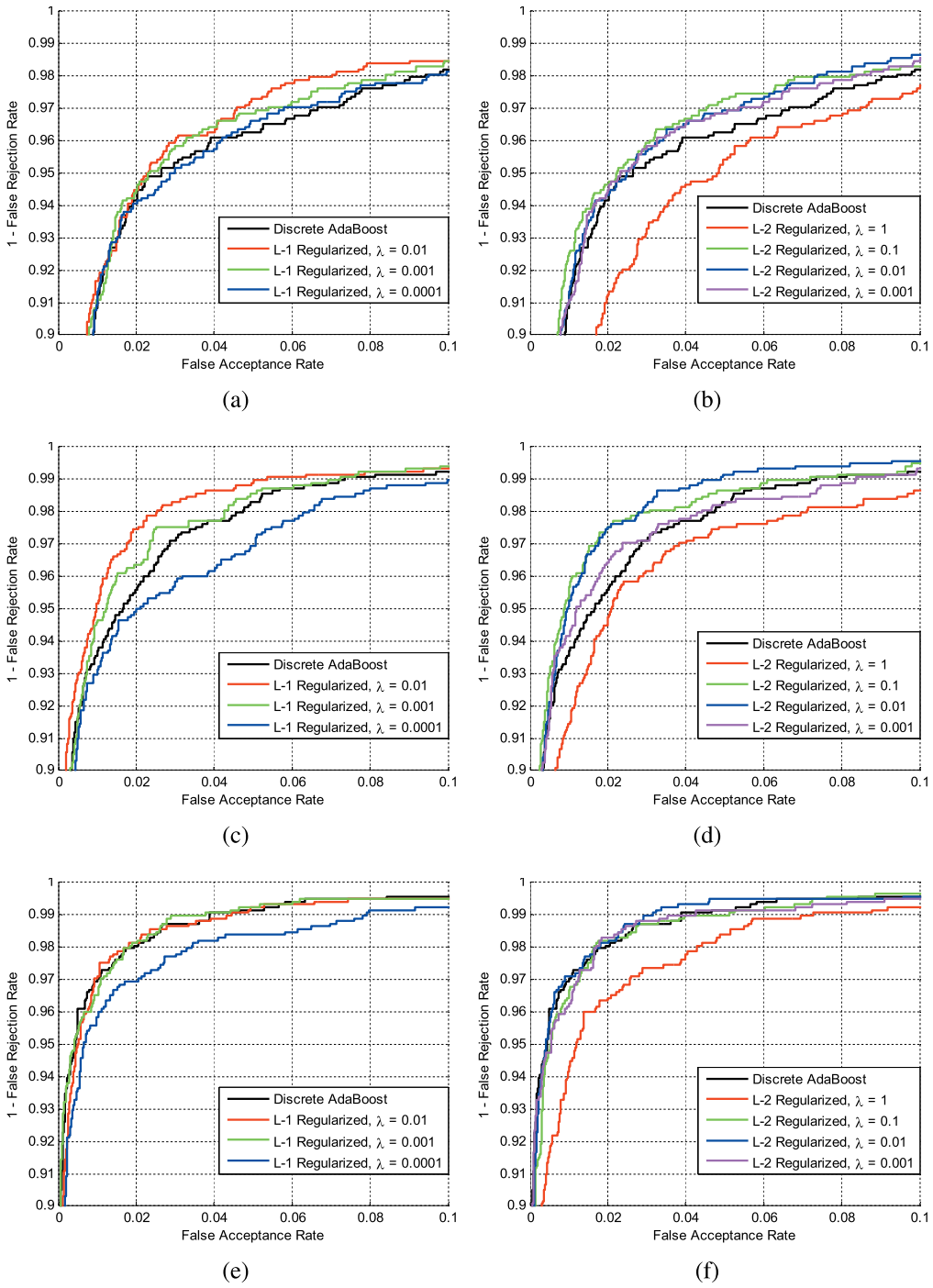


Figure 3.6: ROC curves using (a) L-1 regularization and 25 weak classifiers; (b) L-2 regularization and 25 weak classifiers; (c) L-1 regularization and 50 weak classifiers; (d) L-2 regularization and 50 weak classifiers; (e) L-1 regularization and 100 weak classifiers; and (f) L-2 regularization and 100 weak classifiers.

only compute the weight for the latest introduced weak learner, while weights of those existing weak learners are not changed.

For AdaBoost that uses the exponential loss $l\left(y_i, F\left(\mathbf{x}_i\right)\right) = \exp\left(-y_i F\left(\mathbf{x}_i\right)\right)$, it is clear to see the equivalence between the weak learner selection criterion of totally-corrective AdaBoost and the Discrete AdaBoost, as both can be interpreted as finding the weak learner that best approximates the negative gradient direction of the loss functional, and both selects a binary weak classifier to minimize the loss on the weighted training set in each iteration. However, the functional optimization perspective of the Boosting algorithm also introduces the Real AdaBoost and Gentle AdaBoost, both using continuous-valued weak learners learned as regression functions. Therefore it is interesting to see if plugging the continuous-valued weak learners into a totally-corrective Boosting algorithm will improve the performance.

We select new weak learners using the Real AdaBoost and Gentle AdaBoost method, i.e. learning regression functions of:

$$f\left(\mathbf{x}\right) = \frac{1}{2}\log\frac{P_\omega\left(y=1|\mathbf{x}\right)}{P_\omega\left(y=-1|\mathbf{x}\right)}, \text{ or } f\left(\mathbf{x}\right) = P_\omega\left(y=1|\mathbf{x}\right) - P_\omega\left(y=-1|\mathbf{x}\right),$$

and then solve the totally-corrective AdaBoost optimization problem (3.6) for the selected set of weak learners.

3.3.2 Experimental evaluation

In the following experiment, we consider two types of weak learners, i.e. stumps 3.29 and look up tables (LUT). A LUT for a continuous variable in the range $(-\infty, +\infty)$ is a function of the following form:

$$f\left(x\right) = v_i,\ i = \min\left(n, \max\left(0, \lceil\left(x-b\right)/w\rceil\right)\right), \tag{3.31}$$

If $x \leq b$, it is assigned to bin 0; if $x > b + w \times n$, it is assigned to bin n. The range $[b, b + n \times w]$ is divided into n bins of equal width w. Each bin has a value v_i, and the weak learner score is decided according to which bin a sample instance falls into. We use $n = 64$ for the LUTs, and decide b and w according to the value range of the training instances. For both stump and LUT, the bin values $\{v_i\}$ are decided to minimize the regression error.

We evaluate the methods in the pedestrian-detection problem, using the INRIA dataset. The HOG feature [19] is used. The HOG feature is extracted for 619 blocks

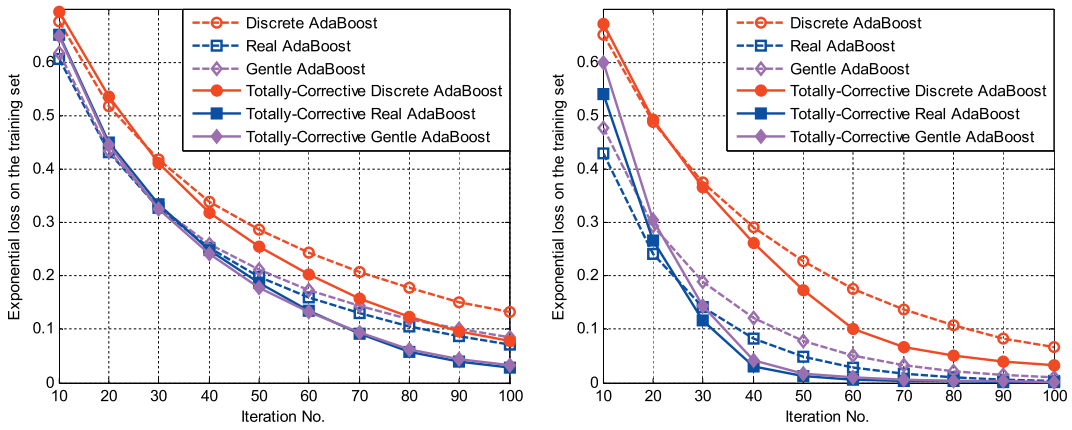


Figure 3.7: (a) Convergence of loss using totally-corrective boosting with stump weak learners. (b) Convergence of loss using totally-corrective boosting with LUT weak learners.

of size 16×16, 32×32 and 64×64, resulting in a feature vector of length 22,284. Then, each weak learner is based on a single feature. The Boosting strong classifier is trained for an intermediate stage of a cascaded detector, where the negative training set consists of fairly difficult samples. 2,474 positive instances and 5,000 negative instances are used to train the classifiers, and then the performance is evaluated using a test set of 1,178 positive instances and 5,000 negative instances. The totally-corrective Boosting problem (3.7) is solved using the MOSEK optimization package.

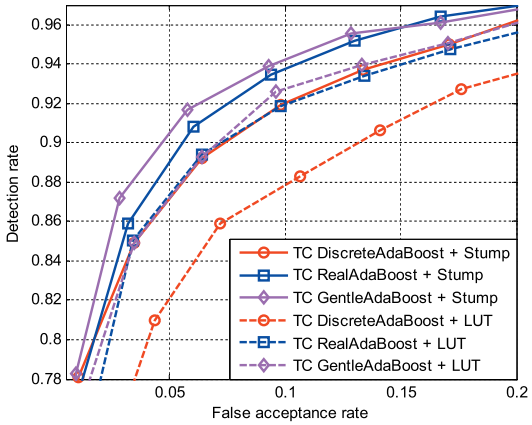


Figure 3.8: ROC curves of totally-corrective AdaBoost with continuous-valued weak learners.

The convergence of exponential loss on the training set and ROC curves on the testing dataset are evaluated. Three conclusions can be drawn from the results. First,

solving the totally-corrective Boosting problem improves performance over the stage-wise forward functional optimization method in terms of empirical loss for Discrete, Real, and Gentle AdaBoost. Second, our intuition that the continuous valued weak learners can also be used in a totally-corrective learning is confirmed by the performance improvement over using binary weak classifiers. Improvement shows on both the training set and the testing set. This is because the 2-nd order information used in the continuous-valued weak learners results in a faster decrease of the empirical loss. Third, as shown in Fig. 3.8, the LUTs show worse generalization performance than the stumps, which underpins the VC theory, since the LUTs have higher VC dimension than stumps.

3.3.3 Conclusions

To conclude, the totally-corrective Boosting is an important advance on the existing Boosting algorithms, and also provides new insights into the nature of the Boosting algorithm generally. Though the column-generation method implies binary classifiers as weak learners, thus emulating the gradient-descent in gradient Boosting, our experiments show that other techniques used in gradient Boosting that outperform gradient descent also lead to better results in combination with totally-corrective updates. In our future research, we hope that our algorithm can be studied in further theoretical depth, and we will also try to apply the hybrid method to other variants of Boosting algorithms.

3.4 Feature subset selection based on partial least squares

In visual object detection, the Boosting algorithm is popular for its embedded feature selection capability. Let each weak learner $f_j(\mathbf{x})$ in $\mathcal{H}$ uses a single feature, e.g. a Haar-like feature or a HOG block, and then the Boosting process selects a subset of weak learners in $\mathcal{H}$ to include in the strong learner, thus selecting a subset of features. In particular, a weak learner is parameterized by both the feature it uses, and its parameters, i.e. $f_j(\mathbf{x}) = f_j(x_j; \theta_j)$. For example, a decision stump is parameterized by a threshold and polarity. In the Boosting iterations, we fit the parameter θ_j according to the weights of the training instances, and select the optimal weak learner. The learning process is as follows:

- Initialize the weights $\{\omega_i\}_{i=1}^M$ of the training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^M$.
- Iterate the following steps:
  1. Train the weak learners $\{f_j\}_{j=1}^N$ on the weighted training set, and select the best weak learner as f_t. Decide its weight w_t.
  2. Add the selected weak learners to the strong learner, i.e. $F \leftarrow F + w_t f_t$, and update the sample weights $\{\omega_i\}_{i=1}^M$.

Fitting the weak learners to the instance-weighted distribution $\{\omega_i\}_{i=1}^M$ is the most computation-intensive part in training a Boosting strong learner, since $\{\omega_i\}_{i=1}^M$ changes in each iteration, and all weak learner need to be retrained. It is highly favorable if we can limit step 1 to a small number of weak learners, i.e. a feature subset. Furthermore, the efficiency gain due to using a reduced subset will diminish if the cost of subset selection is high and the feature subset needs to be refreshed in each Boosting iteration. Therefore our goal is an efficient algorithm to select a feature subset that includes complementary features, from which the Boosting algorithm can train a good strong learner without frequently refreshing the feature subset.

The requirement for complementariness rules out filtering based methods such as measuring the importance of each feature by the correlation or mutual information with the labels, since the features are scored independently and those highly-scored features can be highly correlated with each other. On the other hand, methods that solve a learning problem, e.g. fit a linear function $f(\mathbf{x}; \mathbf{w})$ to predict the labels, may provide a hint about the collaborative importance of the features according to the linear combination weight $\mathbf{w}$, which defines how the features jointly contribute to predicting the labels. Such methods include ridge regression, LASSO regression, logistic regression, SVM, etc. However, another requirement is that the function $f(\mathbf{x}; \mathbf{w})$ can be obtained efficiently. The ridge regression needs to solve the linear system with N variables: $\left(XX^T + \sigma I\right)\mathbf{w} = X\mathbf{y}$, where X is the $N \times M$ matrix of features, $\mathbf{y}$ is the $M \times 1$ vector of labels, and I is a unit matrix. The LASSO regression, logistic regression, and SVM need to be solved using numerical methods. Therefore these methods are not particularly useful for high dimensional feature spaces, since the high computational cost for generating the feature subset cancels out the benefit of using a feature subset.

3.4.1 Partial least square regression

The partial least square regression is an efficient learning algorithm that learns a linear predictor $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, and therefore can be used as a feature subset selection module in Boosting training. The partial least square aims to find latent structures in the feature space and the label space, such that the following two objectives can be achieved. First, the latent structures closely approximate the original feature space; second, the latent structures of the feature space and the label space are highly correlated. Formally, the partial least square regression can be described as follows.

Let $\mathbf{X}$ be the matrix of features, with n rows and N columns, and each row is an instance, each column is a feature. Let $\mathbf{Y}$ be the matrix of labels, with n rows and M columns, and each column is a label. Notice that the partial least square allows for multiple labels, while in our binary classification problem $M = 1$. Therefore in the following we consider $\mathbf{Y} = \mathbf{y}$ to be a vector with values 0 or 1. We want to find a latent structure that approximates the original data using combination of rank-1 matrices as follows:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E} = \mathbf{t}_1\mathbf{p}_1^T + \cdots + \mathbf{t}_K\mathbf{p}_K^T + \mathbf{E}, \tag{3.32}$$

$$\mathbf{y} = \mathbf{U}\mathbf{Q}^T + \mathbf{F} = \mathbf{u}_1 q_1 + \cdots + \mathbf{u}_K q_K + \mathbf{F}. \tag{3.33}$$

$\mathbf{E}$ and $\mathbf{F}$ are the residuals. $\mathbf{t}_i$ and $\mathbf{u}_i$ are called the scores, and $\{\mathbf{p}_i\}$ and $\{q_i\}$ are called the loadings. The latent structure $\{\mathbf{p}_i\}$ and $\{q_i\}$ defines a subspace of the original space (the row space of $\mathbf{X}$ and $\mathbf{y}$). But usually $\{\mathbf{p}_i\}$ or $\{q_i\}$ are not orthogonal and may not be independent, e.g. when that $K > N$ or $K > M$. The original data is represented in the latent structures, using the score matrix $\mathbf{T}$ or $\mathbf{U}$.

The latent structure is constructed in a sequential manner, solving a sequence of problems of the following form:

$$\begin{array}{ll} \max & \operatorname{cov}(\mathbf{t}_k, \mathbf{u}_k) \\ \text{s.t.} & \mathbf{t}_k = \mathbf{X}_k\mathbf{w}_k,\ \mathbf{u}_k = \mathbf{y}_k v_k,\ \|\mathbf{w}_k\|_2 = 1,\ \|v_k\|_2 = 1 \end{array} \tag{3.34}$$

The solution of this problem is that $\mathbf{w}_k$ is the first left-hand singular vector of $\mathbf{X}_k^T\mathbf{y}_k$. For $M = 1$, we have:

$$\mathbf{w}_k = \mathbf{X}_k^T\mathbf{y}_k,\ v_k = 1,\ \mathbf{u}_k = \mathbf{y}_k. \tag{3.35}$$

For further convenience, we normalize $\mathbf{t}_k$ to $\|\mathbf{t}_k\|_2 = 1$ by:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k\|\mathbf{t}_k\|_2, \mathbf{t}_k \leftarrow \mathbf{t}_k/\|\mathbf{t}_k\|_2, \tag{3.36}$$

then $\mathbf{w}_k = \mathbf{X}_k^T\mathbf{y}_k$ still holds. The loadings $\mathbf{p}_k$ and q_k are found to minimize the residual, i.e. solving the following least square problems:

$$\mathbf{X}_k = \mathbf{t}_k\mathbf{p}_k^T \text{ and } \mathbf{y}_k = \mathbf{u}_k q_k. \tag{3.37}$$

The solution of the regression problem is:

$$\mathbf{p}_k = \mathbf{X}_k^T\mathbf{t}_k \text{ and } q_k = \mathbf{y}_k^T\mathbf{u}_k/\mathbf{u}_k^T\mathbf{u}_k \ . \tag{3.38}$$

Then the data is deflated as follows:

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k\mathbf{p}_k^T, \text{ and } \mathbf{y}_{k+1} = \mathbf{y}_k - \mathbf{u}_k q_k. \tag{3.39}$$

It is easy to verify the following properties:

$$\mathbf{t}_k^T\mathbf{X}_{k+i} = 0, \mathbf{t}_k^T\mathbf{t}_{k+i} = 0, \mathbf{X}_{k+i}\mathbf{w}_k = 0, \forall i > 0. \tag{3.40}$$

The partial least square regression assumes that the $\mathbf{y}$ scores $\mathbf{u}_k$ can be predicted from the $\mathbf{X}$ scores $\mathbf{t}_k$ using a linear predictor. This assumption is realistic since the latent structure is found to maximize $\mathrm{cov}\,(\mathbf{t}_k, \mathbf{u}_k)$. The linear predictor c_k is found as follows:

$$\mathbf{u}_k = \mathbf{t}_k c_k \Rightarrow c_k = \mathbf{t}_k^T\mathbf{u}_k, \text{ since } \|\mathbf{t}_k\|_2 = 1. \tag{3.41}$$

Then the approximation $\hat{\mathbf{u}}_k = c_k\mathbf{t}_k$ is used to determine $\hat{q}_k$ and deflate $\mathbf{y}$:

$$\hat{q}_k = \mathbf{y}_k^T\hat{\mathbf{u}}_k/\hat{\mathbf{u}}_k^T\hat{\mathbf{u}}_k \ = \mathbf{y}_k^T\mathbf{t}_k/c_k \ = 1, \tag{3.42}$$

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \hat{\mathbf{u}}_k\hat{q}_k = \mathbf{y}_k - c_k\mathbf{t}_k. \tag{3.43}$$

Finally a linear relation can be found between $\mathbf{X}$ and $\mathbf{y}$:

$$\mathbf{y} = \mathbf{T}\mathbf{c} + \mathbf{F} = \mathbf{X}\mathbf{b} + \mathbf{F}, \tag{3.44}$$

where $\mathbf{b} = \mathbf{W}^*\mathbf{c} = \mathbf{W}^*\mathbf{T}^T\mathbf{y} = \mathbf{W}^*(\mathbf{X}\mathbf{W}^*)^T\mathbf{y}$ is the linear coefficient for predicting

$\mathbf{y}$ from $\mathbf{X}$, and $\mathbf{W}^*$ is the matrix for calculating $\mathbf{T}$ from $\mathbf{X}$:

$$\mathbf{T} = \mathbf{X}\mathbf{W}^*. \tag{3.45}$$

$\mathbf{W}^*$ can be derived as follows:

$$\begin{aligned}
&\because \mathbf{X}\mathbf{w}_k = \left(\mathbf{T}\mathbf{P}^T + \mathbf{X}_{K+1}\right)\mathbf{w}_k = \mathbf{T}\mathbf{P}^T\mathbf{w}_k, k = 1, ..., K, \\
&\therefore \mathbf{X}\mathbf{W} = \mathbf{T}\mathbf{P}^T\mathbf{W} \\
&\quad \Rightarrow \mathbf{T} = \mathbf{X}\mathbf{W}\left(\mathbf{P}^T\mathbf{W}\right)^{-1} \\
&\quad \Rightarrow \mathbf{W}^* = \mathbf{W}\left(\mathbf{P}^T\mathbf{W}\right)^{-1}
\end{aligned} \tag{3.46}$$

And finally we have the linear predictor between $\mathbf{X}$ and $\mathbf{y}$, represented by:

$$\mathbf{b} = \mathbf{W}^*\mathbf{c} = \mathbf{W}^*\mathbf{T}^T\mathbf{y} = \mathbf{W}^*(\mathbf{X}\mathbf{W}^*)^T\mathbf{y} = \mathbf{W}\left(\mathbf{P}^T\mathbf{W}\right)^{-1}\left(\mathbf{W}^T\mathbf{P}\right)^{-1}\mathbf{W}^T\mathbf{X}^T\mathbf{y}. \tag{3.47}$$

The complexity of the partial least square regression is approximately $O(nNK)$, where n is the number of training instances, N is the number of features, and K is the order of the latent structure. Usually K is much smaller than n or N, for example, $K = 20 \sim 30$ is sufficient in our test. The partial least square regression for $M = 1$ is summarized in Algorithm 2.

Algorithm 2 Partial least square regression for scalar labels

Initialization: $\mathbf{X}_1 = \mathbf{X}$, $\mathbf{y}_1 = \mathbf{y}$.
for $k = 1$ to K **do**
  Calculate the following terms:
  (1) $\mathbf{w}_k = \mathbf{X}_k^T\mathbf{y}_k$, $\mathbf{t}_k = \mathbf{X}_k\mathbf{w}_k$
  (2) $\mathbf{w}_k \leftarrow \mathbf{w}_k\|\mathbf{t}_k\|_2$, $\mathbf{t}_k \leftarrow \mathbf{t}_k/\|\mathbf{t}_k\|_2$
  (3) $\mathbf{p}_k = \mathbf{X}_k^T\mathbf{t}_k$
  (4) $c_k = \mathbf{t}_k^T\mathbf{u}_k = \mathbf{t}_k^T\mathbf{y}_k$
  (5) $\mathbf{y}_{k+1} = \mathbf{y}_k - \mathbf{t}_k c_k$, $\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{t}_k\mathbf{p}_k^T$
end for
Calculate $\mathbf{b}$:
  $\mathbf{W}^* = \mathbf{W}\left(\mathbf{P}^T\mathbf{W}\right)^{-1}$
  $\mathbf{b} = \mathbf{W}^*\mathbf{c} = \mathbf{W}^*\mathbf{T}^T\mathbf{y} = \mathbf{W}^*(\mathbf{X}\mathbf{W}^*)^T\mathbf{y}$
return $\mathbf{b}$.

3.4.2 PLS for feature subset selection

The values in the linear coefficient vector $\mathbf{b}$ reflect how each feature contributes to predicting the label $\mathbf{y}$ in a collaborative manner. Therefore a feature selection scheme can be implemented by treating the values in $\mathbf{b}$ as importance scores. However, the weights of those highly correlated features will be scaled down. Consider a highly relevant feature that has several duplicated copies in the feature vector $\mathbf{x}$. The copies of this feature will be equally weighted in $\mathbf{b}$, while the sum of their weights will equal to that of the feature without duplication. To cope with this situation, instead of selecting those highest weighted features, we treat $\mathbf{b}$ as defining an importance distribution, and draw features from this distribution.

Another issue for using PLS for feature selection in Boosting is how to perform PLS on weighted training instances with the weight $\boldsymbol{\omega}$. An unweighted dataset can be obtained by resampling the weighted dataset, using importance sampling. Learning on this resampled dataset is equivalent to learning on the original weighted dataset. In the following $\sim$ stands for working with the resampled dataset. Finding the latent structure for the resampled dataset can be described by the following problem:

$$\begin{array}{ll} \max & \operatorname{cov}\left(\tilde{\mathbf{t}}, \tilde{\mathbf{u}}\right) \\ \text{s.t.} & \tilde{\mathbf{t}} = \tilde{\mathbf{X}}\tilde{\mathbf{w}},\ \tilde{\mathbf{u}} = \tilde{\mathbf{y}}\tilde{v},\ \|\tilde{\mathbf{w}}\|_2 = 1,\ \|\tilde{v}\|_2 = 1. \end{array} \tag{3.48}$$

And we consider the following problem absorbing the weights into the training data:

$$\begin{array}{ll} \max & \operatorname{cov}\left(\mathbf{t}_{nn}, \mathbf{u}_{nn}\right) \\ \text{s.t.} & \mathbf{t}_{nn} = \mathbf{X}_{nn}\mathbf{w}_{nn},\ \mathbf{u}_{nn} = \mathbf{y}_{nn}v_{nn},\ \|\mathbf{w}_{nn}\|_2 = 1,\ \|v_{nn}\|_2 = 1, \end{array} \tag{3.49}$$

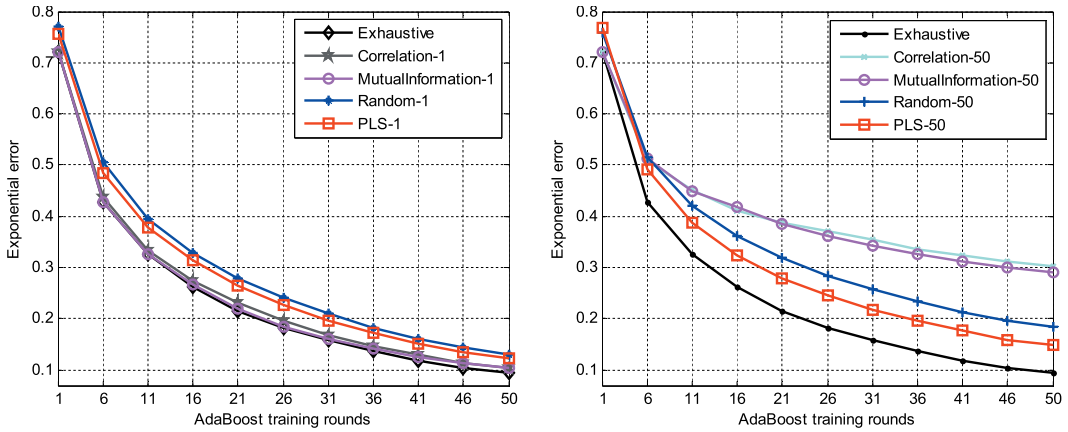
where $\mathbf{X}_{nn} = diag\left(\sqrt{\boldsymbol{\omega}}\right)\mathbf{X}$, and $\mathbf{y}_{nn} = diag\left(\sqrt{\boldsymbol{\omega}}\right)\mathbf{y}$. It is easy to see that $\tilde{\mathbf{w}} = \mathbf{w}_{nn}$ since $\mathbf{X}_{nn}^T\mathbf{y}_{nn} = \mathbf{X}^T diag\left(\omega\right)\mathbf{y} = \tilde{\mathbf{X}}^T\tilde{\mathbf{y}}$. Therefore:

$$\begin{aligned} &\tilde{\mathbf{t}} = \tilde{\mathbf{X}}\tilde{\mathbf{w}} = \tilde{\mathbf{X}}\mathbf{w}_{nn}, \\ &\tilde{\mathbf{p}} = \tilde{\mathbf{X}}^T\tilde{\mathbf{t}} = \tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\mathbf{w}_{nn} = \mathbf{X}_{nn}^T\mathbf{X}_{nn}\mathbf{w}_{nn} = \mathbf{p}_{nn}. \end{aligned} \tag{3.50}$$

The conclusion is that $\mathbf{b}_{nn} = \tilde{\mathbf{b}}$. Therefore we solve the PLS regression problem for $(\mathbf{X}_{nn}, \mathbf{y}_{nn})$, and then select a subset of features according to $\mathbf{b}_{nn}$.

3.4.3 Experimental evaluation

We test the PLS feature subset selection algorithm on the INRIA pedestrian dataset, using 2,474 training instances and 8,000 negative training instances. The negative training set includes 4,000 windows randomly selected from the negative images, and 4,000 hard negatives collected by a linear SVM detector using HOG+LBP features. The full feature set includes 25,768 HOG features obtained by concatenating the HOG from 688 rectangular blocks. The Discrete AdaBoost algorithm is used for training the classifier, using decision stumps as weak classifiers. We compare the PLS feature subset selection algorithm with several baseline methods, including exhaustively testing all features, filtering based methods using correlation and mutual information, and random selection. A feature subset of size 500 (2% of the full dataset) is selected, from which the best feature is then selected by AdaBoost. To avoid extracting all features for the whole training set, a subset of the training instances is used for feature selection, using 1,000 positive instances and 2,000 negative instances. Experiments show that this does not impair the quality of the selected feature subset. To evaluate the performance, we show the empirical loss plotted against the number of Boosting iterations.



(a) (b)

Figure 3.9: (a) Comparison of empirical loss convergence using various subset selection method, update period = 1. (b) Comparison of empirical loss convergence using various subset selection method, update period = 50.

First we update the feature subset in every iteration, and the results are shown in Fig. 3.9a. The correlation and mutual information methods have very high probability to include the best feature, and the performances are close to exhaustively testing

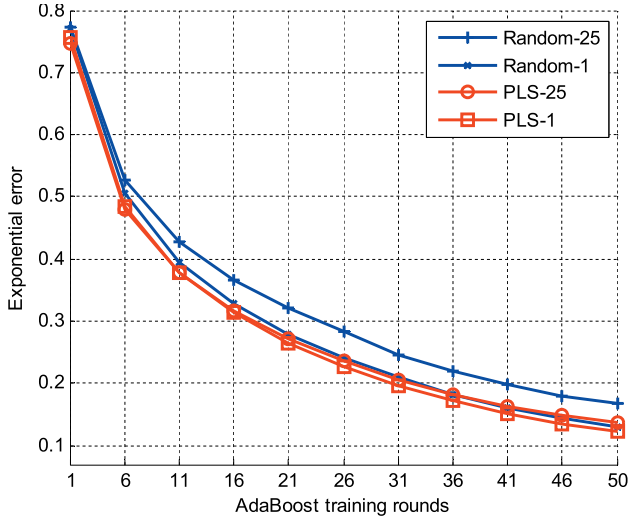


Figure 3.10: Comparison of empirical loss convergence using the PLS selected subsets and random subsets.

all the features. The PLS subset selection is better than the randomly selected subset, showing that the chance of including useful features is improved by sampling from a weighted distribution with the weights calculated by PLS. Then we refresh the feature subset every 50 iterations, i.e. 50 weak classifiers are to be trained from the feature subset. The results are shown in Fig. 3.9b The correlation and mutual information perform poorly in this case, showing that the highest scored features are also highly correlated with each other, hence the feature subset fails to include diverse and complementary features. The advantage of PLS feature subset selection over the other methods is obvious, including the random subset. This indicates that the PLS method generates a much more informative feature subset than random selection, and also includes diverse and complementary features, in contrast to the correlation and mutual information methods. Finally in Fig. in Fig. 3.10 we compare the PLS subset with the randomly selected subset. The figure shows that a PLS subset updated every 25 iterations performs similarly to a randomly selected subset that is updated in every iteration (Random-1). Considering that the random subset does not incur any additional cost in feature selection, this seems diminish the benefit of the subset selection algorithm. However, over the 50 Boosting iterations, the Random-1 scheme explores almost the entire set of features for the training set, such that all the features need to be extracted for the training set, while the PLS-25 scheme only needs to extract the full feature set for the small dataset used for feature selection, and only the selected feature subset, i.e. at most 1,000 features, need to be extracted for the much larger training dataset. The advantage of the feature subset selection algorithm is also in the reduction of feature

extraction and feature storage for the training data.

3.4.4 Conclusions

To conclude, we have applied the feature-selection techniques for selecting working feature subsets in AdaBoost training. Based on the partial least square regression approach and the novel sampling-based feature selection approach, an effective feature subset can be selected, with low computational overheads. The training efficiency of the AdaBoost algorithm can be improved by the proposed approach.

For the future work, the proposed approach can be tested with other weak classifiers and other variants of AdaBoost. The proposed approach can also be embedded in a fully functional pattern recognition system in order to evaluate the improvement at the system level.

Chapter 4 Boosting cascade detectors

Several issues involved in designing a Boosting cascade detector are discussed in this chapter. To begin with, we give a short introduction of the cascade, pointing out the problems that can be solved by a cascade in Section 4.1. Then three questions arising from training a cascade are studied. First, each stage of a cascade detector is usually makes a highly unbalanced decision, accepting almost all the positive instances while rejecting an appropriate portion of the negatives. Therefore training an optimal classifier for this unbalanced objective is studied in Section 4.2. Then in Section 4.3, we discuss how to set the goal for each stage of the cascade, such that the cascade is efficient as a whole. Lastly, in Section 4.4, how to re-use information in the early stages of the cascade is studied, and we propose a biased selection strategy for re-using weak classifiers and features that are already obtained, such that the detector efficiency is further improved.

4.1 Cascade detectors

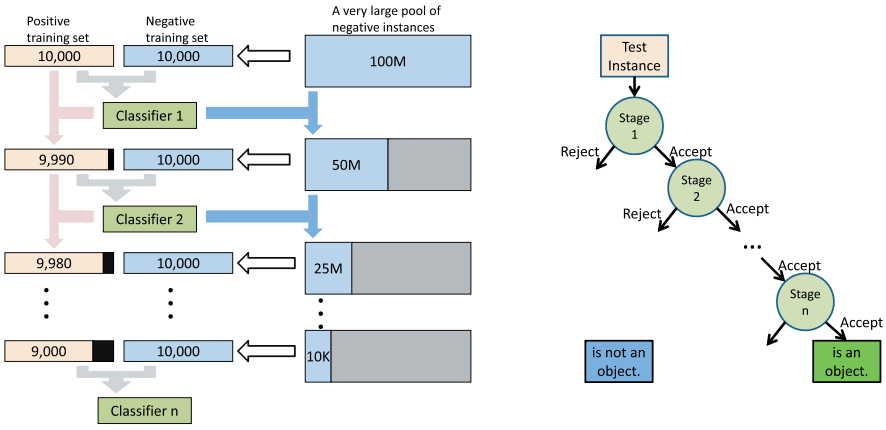
The cascade approach is popular since it solves three problems in sliding window style visual object detection. First, there are numerous non-object windows while only a small number of object windows in typical images, hence the detector should have a very low false positive rate but an acceptable detection rate. Second, we usually have limited number of positive training instances, while numerous negative training instances can be collected. How to efficiently exploit the negative training data is a problem for the learning process. Third, the large number of windows to evaluate in a sliding window scheme incurs high computational cost in detection, for which a highly efficient detector is expected.

While the detector efficiency and performance are often contradictory goals, the cascade approach achieves both by chaining a number of classifiers, each called a stage of the cascade. Only those instances accepted by an earlier stage will be passed to a later stage. Assume each stage classifier has recall rate a, and false positive rate b, by chaining up n stages, we obtain a detector with recall rate a^n and false-positive rate b^n. It is easy to train a detector with very high recall and moderate false positive

rate, e.g. $a = 0.995$ and $b = 0.5$; then 20 stages aggregate into a detector with recall $a^n = 0.995^{20} \approx 0.90$ and false positive rate $b^n = 0.5^{20} \approx 10^{-6}$. Assume that each stage has a fixed computational cost c. Since the majority of windows are non-objects, the computational cost per-window can be calculated as:

$$c \sum_{i=0}^{n-1} b^i = c \frac{(1 - b^n)}{1 - b} \approx \frac{c}{1 - b}.$$

Therefore, the expected cost of the whole cascade is on the same order as the cost of a single stage, which indicates that the cascade is a very efficient detector. Besides, the cascade is usually designed such that the earlier stages are very simple; therefore most windows can be rejected with a very small amount of computation, further enhancing the detector efficiency. As for exploiting the huge negative training set, the cascade stages are usually trained sequentially, such that only the false positives of the earlier stages are used for training the later stages. Therefore the size of the remaining negative dataset quickly decreases, and the full negative dataset can be effectively exploited. The training and testing of a cascade detector are illustrated in Fig. 4.1.



(a) (b)

Figure 4.1: Illustration of the training (a) and testing (b) of a cascade detector.

The Boosting algorithm is very popular in combination with the cascade approach, since it builds efficient classifiers with feature selection, and offers explicit control of the complexity of each stage, such that we use just enough weak classifiers (features) to achieve the desired recall and false positive rate (a, b) at each stage. In this section we study several issues in designing a boosting cascade detector. In Section 4.2, we study how to learn a classifier that is aware of the asymmetric training goal (a, b),

achieving this goal with minimal classifier complexity. Then in Section 4.3 we study how to set the goal (a, b) for each stage, such that the detector complexity is minimized as a whole. Finally in Section 4.4, recycling the features and weak classifiers between stages and within a stage is studied, such that the complexity of the detector can be effectively reduced without impairing the performance.

4.2 Learning with asymmetric goals

The false rejection rate (FRR), false acceptance rate (FAR), and error rate (ER) of a binary classifier are defined as follows:

$$FRR = \frac{FN}{TP + FN}, FAR = \frac{FP}{FP + TN}, ER = \frac{FP + FN}{TP + FN + FP + TN}. \quad (4.1)$$

		Predicted	
		Positive	Negative
Ground truth	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Each stage of the cascade is expected to have very low FRR but moderate FAR. The Boosting algorithm sequentially adds new weak learners to the classifier, and pushes the ROC curve upwards until the desired FRR and FAR are simultaneously achieved. However, as we have discussed before, the learning algorithms minimize an upper bound of ER rather than the asymmetric FRR/FAR goal. Therefore, the classifier could be sub-optimal, for example, using more weak classifiers than necessary. In the following, we introduce methods that address the asymmetric learning goal. First we give a naive formulation based on training on weighted datasets. Then, the cost sensitive Bayes decision rule [71] is introduced, and a discriminative approach for approximating the cost sensitive boundary is formulated. Lastly we discuss methods that directly optimize the FRR and FAR, i.e. the linear asymmetric learning [69] and LACBoost [70].

4.2.1 Naive weight manipulation

The cost-sensitive learning is considered as a proxy to achieve the asymmetric FRR and FAR. Let each false negative elicits cost C_1, and each false positive elicits cost

C_{-1}, then by tuning the cost parameters, we can make a classifier that minimizes the cost-sensitive loss:

$$\min_{f} \quad \int_{\mathbf{x}} C_1 p\left(f \neq y, \mathbf{x}, y = 1\right) + C_{-1} p\left(f \neq y, \mathbf{x}, y = -1\right) d\mathbf{x} \tag{4.2}$$

Assume the loss is evenly split between the positive and the negative data, i.e.

$$\int_{\mathbf{x}} C_1 p\left(f \neq y, \mathbf{x}, y = 1\right) d\mathbf{x} = \int_{\mathbf{x}} C_{-1} p\left(f \neq y, \mathbf{x}, y = -1\right) d\mathbf{x}, \tag{4.3}$$

and assume equal prior probability of the two classes, i.e. $P\left(y = 1\right) = P\left(y = -1\right)$, then we can obtain:

$$C_1 FAR = C_{-1} FRR, \text{ i.e. } \frac{FRR}{FAR} = \frac{C_{-1}}{C_1}. \tag{4.4}$$

Then we can expect that by setting C_1 and C_{-1} according to the target FRR and FAR, the classifier will be optimal for achieiving the asymmetric goal, e.g. using the least number of weak classifiers. However, in practice, the empirical loss used in the learning problem (e.g. exponential loss) is only an approximation of the 0-1 loss, and the ratio of the empirical loss on the positive and negative sets usually is not equal to the ratio between FRR and FAR. We show the exponential loss and error rates on the positive and negative training set during training a Discrete AdaBoost classifier in Fig. 4.2, for $C_{-1} = 1$ and $C_1 = 1$, 2, and 5. We can see the ratio of the exponential loss closely follows the specified C_1 and C_{-1}, but the ratio of the empirical loss fails to predict the ratio of 0-1 loss, i.e. the ratio of FRR and FAR. However, the cost sensitive loss (4.2) is still of interest, and is a a good hint about how we should deal with the asymmetric learning goal.

4.2.2 Optimal cost sensitive decision rule

Assume we have the true posterior probabilities $P\left(y = 1|\mathbf{x}\right)$ and $P\left(y = -1|\mathbf{x}\right)$. Then an optimal cost-insensitive classifier is

$$h\left(\mathbf{x}\right) = sign\left(f\left(\mathbf{x}\right)\right), \text{ where } f\left(\mathbf{x}\right) = \log \frac{P\left(y = 1|\mathbf{x}\right)}{P\left(y = -1|\mathbf{x}\right)}. \tag{4.5}$$

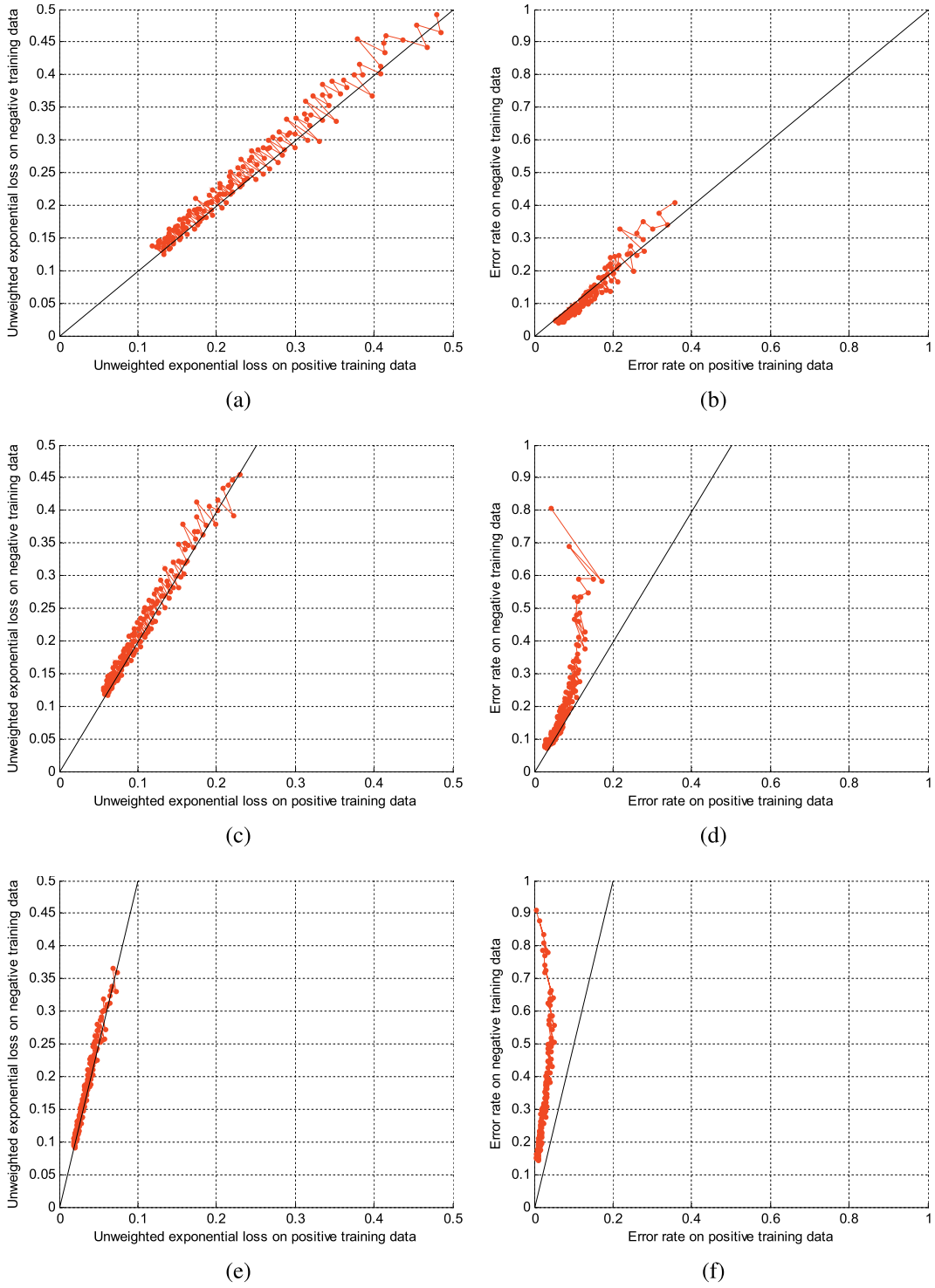
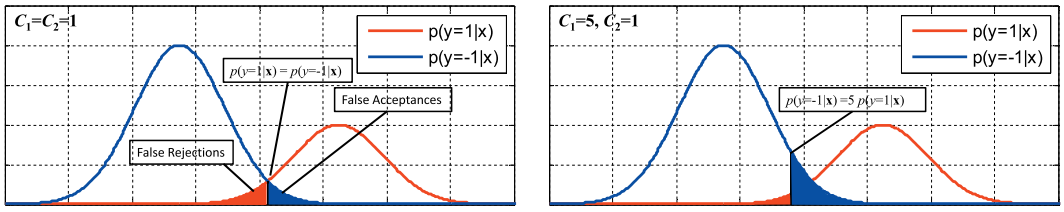


Figure 4.2: (a) (b) Exponential loss and 0-1 loss on the training set, for $C_1 = 1$, $C_2 = 1$. (c) (d) Exponential loss and 0-1 loss on the training set, for $C_1 = 2$, $C_2 = 1$. (e) (f) Exponential loss and 0-1 loss on the training set, for $C_1 = 5$, $C_2 = 1$.

And the optimal cost-sensitive scoring function is:

$$f^{*}(\mathbf{x}) = \log \frac{C_1 P(y=1|\mathbf{x})}{C_{-1} P(y=-1|\mathbf{x})} = f(\mathbf{x}) + \log \frac{C_1}{C_{-1}}. \tag{4.6}$$

This suggests that a cost-sensitive classifier can be obtained by simply adjusting the threshold of the cost-insensitive score function $f(\mathbf{x})$, if $f(\mathbf{x})$ is an accurate predictor of the log odds everywhere. This is also the most popular technique in practice, and we simply pick a proper operating point from the ROC curve derived from $f(\mathbf{x})$. The optimal decision boundaries for cost insensitive and cost sensitive loss with $C_1 = 5$ and $C_2 = 1$ are illustrated in Fig. 4.3 for a 1-D problem. Notice that the ratio of cost factors does not directly imply the ratio of errors (i.e. the area of the red and blue regions).



(a) (b)

Figure 4.3: Illustration of the Bayes optimal decision boundary for (a) $C_1 = 1, C_2 = 1$; (b) $C_1 = 5, C_2 = 1$.

However, the solution of a cost-insensitive learning problem usually only considers approximating the separating boundary, i.e. the 0 level set of $f(\mathbf{x})$ approximates the 0 level set of $\log \frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})}$:

$$\left\{\mathbf{x} : \hat{f}(\mathbf{x}) = 0\right\} \approx \left\{\mathbf{x} : \log \frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} = 0\right\}. \tag{4.7}$$

But no guarantee can be made for the approximation at other locations, i.e. there may not exist a level set of $f(\mathbf{x})$ that closely approximates $\left\{\mathbf{x} : \log \frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} = \log \frac{C_{-1}}{C_1}\right\}$. Therefore adjusting the threshold cannot guarantee an optimal cost sensitive decision boundary. Furthermore, Boosting selects and combines weak learners; the weak learners selected according to the symmetric cost may also be sub-optimal for the asymmetric cost.

4.2.3 Cost-sensitive loss functions

The learner should produce a scoring function $f(\mathbf{x})$ whose 0 level set directly approximates the cost sensitive Bayes decision boundary (4.5). It has been shown that the minimizer of the exponential loss functional is the optimal cost-insensitive Bayes decision boundary:

$$\arg\min_f E_D\left(\exp\left(-yf\left(\mathbf{x}\right)\right)\right) = \frac{1}{2}\log\frac{P_D\left(y=1|\mathbf{x}\right)}{P_D\left(y=-1|\mathbf{x}\right)}, \tag{4.8}$$

where D is the distribution over $(\mathbf{x}, y)$. Similarly, we can prove that the minimizer of the following naively weighted loss is the Bayes optimal decision boundary for the cost sensitive loss:

$$L\left(y, f\left(\mathbf{x}\right)\right) = C_y \exp\left(-yf\left(\mathbf{x}\right)\right), \tag{4.9}$$

where $y \in \{+1, -1\}$. The minimizer is:

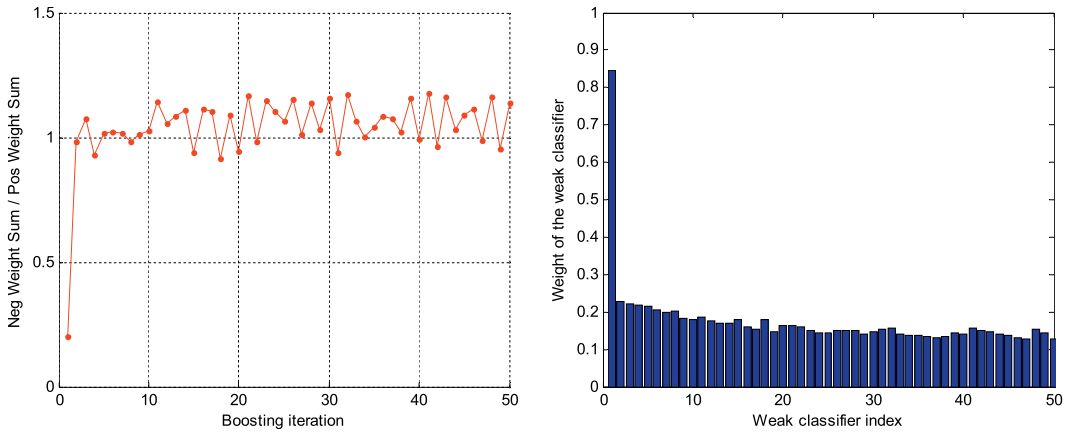
$$\arg\min_f E_D\left(L\left(y, f\left(\mathbf{x}\right)\right)\right) = \frac{1}{2}\log\frac{C_1 P_D\left(y=1|\mathbf{x}\right)}{C_{-1} P_D\left(y=-1|\mathbf{x}\right)} \tag{4.10}$$

However, this naive weight adjustment has been criticized by many works. For example, using Discrete AdaBoost with loss (4.9), the training instances are initially weighted such that each positive instance has weight C_1 and each negative instance has weight C_{-1}, creating an asymmetric distribution over the two classes. But in practice the asymmetry is immediately destroyed after obtaining the first weak classifier, such that only the first weak classifier is actually trained on an asymmetric data. We show an example by training a Discrete AdaBoost classifier using $C_1 = 5$ and $C_{-1} = 1$. The ratio between the sum of positive instance weights and the sum of negative instance weights are shown in Fig. 4.4a, and the weak classifier weights are shown in Fig. 4.4b. It can be observed that only the first weak classifier is trained on an asymmetric distribution, and the first weak classifier is highly weighted for it contributes most to minimizing the cost sensitive loss.

Let $\eta\left(\mathbf{x}\right) = P_D\left(y=1|\mathbf{x}\right)$, [71] studies cost-sensitive learning by studying the conditional risk:

$$C\left(\eta\right) = \eta C_1 I\left(f\left(\mathbf{x}\right) \leq 0\right) + \left(1-\eta\right) C_{-1} I\left(f\left(\mathbf{x}\right) \geq 0\right), \tag{4.11}$$

where I is the indicator function and measures the 0-1 loss. The optimal predictor f



(a) (b)

Figure 4.4: (a) Ratio of the sum of positive instance weights and sum of negative weights for training each weak classifier; (b) Weak classifier weights in the Discrete AdaBoost classifier.

is:

$$\begin{cases} f(\mathbf{x}) > 0, & \text{if } \eta(\mathbf{x}) > \frac{C_{-1}}{C_1+C_{-1}} \\ f(\mathbf{x}) = 0, & \text{if } \eta(\mathbf{x}) = \frac{C_{-1}}{C_1+C_{-1}} \\ f(\mathbf{x}) < 0, & \text{if } \eta(\mathbf{x}) < \frac{C_{-1}}{C_1+C_{-1}} \end{cases} \tag{4.12}$$

Two properties are satisfied by (4.11), which uses the 0-1 loss:

1. The risk (4.11) is maximized at $\eta^* = \frac{C_{-1}}{C_1+C_{-1}}$.
2. The risk is symmetric with respect to the cost factors C_1 and C_{-1} in a neighborhood of $\eta^* = \frac{C_{-1}}{C_1+C_{-1}}$:

$$C(\eta^* - \varepsilon C_{-1}) = C(\eta^* + \varepsilon C_1). \tag{4.13}$$

Replacing the 0-1 loss with an empirical loss function, e.g. the exponential loss, the loss function for cost-sensitive problems is expected to also satisfy the two properties, in addition to approximating the optimal cost sensitive Bayes decision boundary. By replacing the 0-1 loss with the naively weighted exponential loss (4.9), and substituting the solution (4.10), we can find that the conditional risk is:

$$C(\eta) = \eta C_1 \sqrt{\frac{C_{-1}(1-\eta)}{C_1\eta}} + (1-\eta) C_{-1} \sqrt{\frac{C_1\eta}{C_{-1}(1-\eta)}} \tag{4.14}$$

The risk (4.14) is maximized at $\eta = 1/2$, which violates property 1. Therefore [71]

proposes another form of cost-sensitive exponential loss:

$$L_C\left(y, f\left(\mathbf{x}\right)\right) = \exp\left(-C_y y f\left(\mathbf{x}\right)\right)$$

which is minimized at:

$$\arg\min_f E_D\left(L_C\left(y, f\left(\mathbf{x}\right)\right)\right) = \frac{1}{C_1 + C_{-1}} \log \frac{C_1 P_D\left(y = 1|\mathbf{x}\right)}{C_{-1} P_D\left(y = -1|\mathbf{x}\right)}, \tag{4.15}$$

whose 0 level set is also the Bayes optimal cost sensitive decision boundary. In addition, the scoring function (4.15) satisfies property 1. (4.15) does not scale the loss as in (4.9); instead, the scoring function $f\left(\mathbf{x}\right)$ is scaled before calculating the loss, such that a positive instance and a negative instance on the decision boundary $f\left(\mathbf{x}\right) = 0$ incur the same amount of loss.

Cost sensitive Boosting algorithms can be derived for the cost sensitive loss (4.15). The original AdaBoost algorithms should be modified as follows. First, the weight update formula of the training instances is changed to $\omega_i = C_{y_i} e^{-C_{y_i} y_i F(\mathbf{x}_i)}$. Then corresponding to the three methods in section 3.2 that solve the AdaBoost as a functional optimization problem, the algorithms for cost-sensitive loss can be described as follows:

Cost sensitive Discrete AdaBoost Select the new weak classifier to minimize the weighted error as before. The weight of the new weak classifier cannot be obtained in closed form, and a line search is needed to find the optimal weight for the new weak classifier.

Cost sensitive Real AdaBoost The new weak classifier is the regression function to approximate the following target value:

$$f_j\left(\mathbf{x}\right) = \frac{1}{C_1 + C_{-1}} \log \frac{P_\omega\left(y = 1|\mathbf{x}\right)}{P_\omega\left(y = -1|\mathbf{x}\right)}. \tag{4.16}$$

Cost sensitive Gentle AdaBoost The new weak classifier is the regression function to approximate the following target value:

$$f_j\left(x\right) = \frac{P_\omega\left(y = 1, x\right) - P_\omega\left(y = -1, x\right)}{C_1 P_\omega\left(y = 1, x\right) + C_{-1} P_\omega\left(y = -1, x\right)}. \tag{4.17}$$

Now we illustrate the cost-sensitive learning using artificial data. The positive data is drawn from a 2-D Gaussian with $\sigma = 1$, centered on $(0, 0)$. The distribution of the

negative data is centered on a ring with radius 3.5, and with standard deviation 1 in the profile direction of the ring. We evaluate the Gentle AdaBoost, training the strong classifier up to 40 weak classifiers using regression stumps as weak classifiers. Fig 4.5a shows the groundtruth cost-insensitive decision boundary and the 0 level set of the learned classifier, and we can see the 0 level set closely approximates the groundtruth decision boundary. Then the 0-levelsets of the classifiers using the naive weight manipulation (4.9) and the modified loss (4.15) are compared, for $(C_1 = 2, C_{-1} = 1)$ and $(C_1 = 5, C_{-1} = 1)$, and the results are shown in Fig. 4.5b to 4.5e. We can see indeed both approaches are effective in approximating the cost sensitive optimal decision boundary.

4.2.4 Direct formulation of the asymmetric goal

The cost sensitive learning optimizes for the cost sensitive decision boundary. However, the cost of each type of error does not directly imply the ratio of FNR and FPR. Directly optimizing for the recall and FAR are studied in [69, 70], using the following learning problem:

$$\begin{array}{ll} \min\limits_{f} & FNR = \int_{\mathcal{X}} p\left(f \neq y, \mathbf{x} | y = 1\right) d\mathbf{x}, \\ \text{s.t.} & FPR = \int_{\mathcal{X}} p\left(f \neq y, \mathbf{x} | y = -1\right) d\mathbf{x} = const. \end{array} \tag{4.18}$$

i.e. minimizing the FNR for a particular choice of FPR. The problem is usually difficult to solve. However, if $FPR = 0.5$ and the classifier is linear, i.e. $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the problem can be written as:

$$\begin{array}{ll} \max\limits_{\mathbf{w}\neq 0, b} & P_{D+}\left(\mathbf{w}^T\mathbf{x} \geq b | y = 1\right), \\ \text{s.t.} & P_{D-}\left(\mathbf{w}^T\mathbf{x} \leq b | y = -1\right) = 0.5, \end{array} \tag{4.19}$$

where $D+$ and $D-$ stand for the conditional distribution of the positive and negative class, respectively. If $\mathbf{w}^T\mathbf{x}$ is symmetric for the negative class, the solution can be obtained by solving the following problem:

$$\max_{\mathbf{w}} \frac{\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)}{\sqrt{\mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w}}}, \tag{4.20}$$

where $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_{-1}$ are the mean of the two classes, respectively; $\boldsymbol{\Sigma}_1$ is the covariance matrix of the positive class. To derive the formulation of problem (4.20), the following

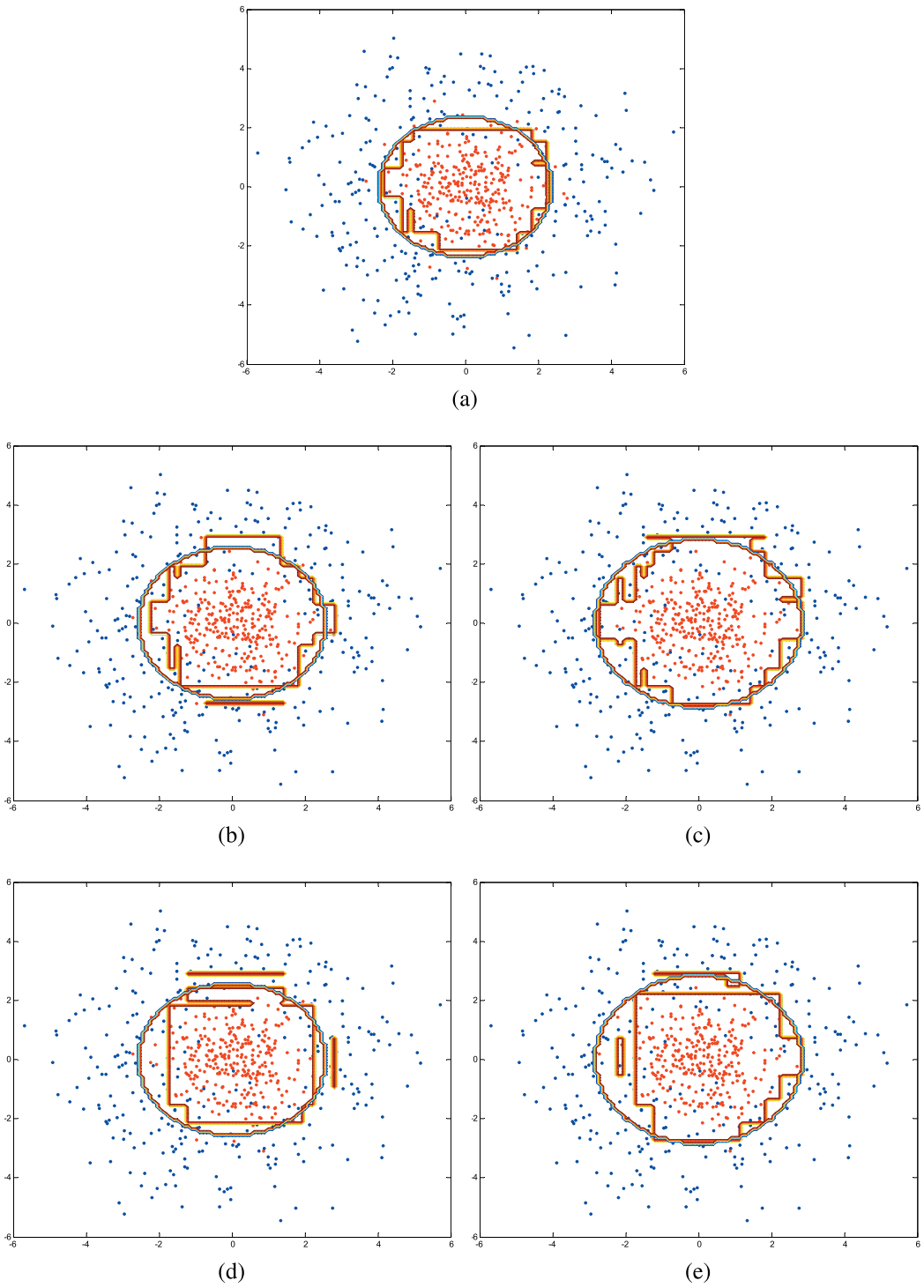


Figure 4.5: 0-levelset of the Gentle AdaBoost decision boundary and the groundtruth optimal decision boundary. (a) using symmetric cost; (b) and (c) using the naive weight manipulation (4.9) for $(C_1 = 2, C_{-1} = 1)$ and $(C_1 = 5, C_{-1} = 1)$; (d) and (e) using the modified loss (4.15) for $(C_1 = 2, C_{-1} = 1)$ and $(C_1 = 5, C_{-1} = 1)$.

problem is studied:

$$\begin{aligned} \max_{\mathbf{w}\neq 0, b} \quad & \gamma \\ \text{s.t.} \quad & \inf_{D+} P_{D+}\left(\mathbf{w}^T\mathbf{x} \geq b\right) \geq \gamma, \\ & \inf_{D-} P_{D-}\left(\mathbf{w}^T\mathbf{x} \leq b\right) \geq 0.5. \end{aligned} \tag{4.21}$$

Notice that the infimum in (4.21) are over all possible distributions $D-$ and $D+$. If $D-$ is a symmetric distribution, obviously $P_{D-}\left(\mathbf{w}^T\mathbf{x} \leq b\right) \geq 0.5$ is satisfied by $b \geq \mathbf{w}^T\boldsymbol{\mu}_{-1}$, and we can replace the second constraint in (4.21) by $b \geq \mathbf{w}^T\boldsymbol{\mu}_{-1}$. To derive the solution to problem (4.21), we study the following problem:

$$\inf_{D+} P_{D+}\left(\mathbf{w}^T\mathbf{x} \geq b\right) \geq \gamma. \tag{4.22}$$

Here we directly give the solution from [70]:

- If $D_+ = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is an arbitrary distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$b \geq \mathbf{w}^T\boldsymbol{\mu} + \sqrt{\frac{\gamma}{1-\gamma}} \cdot \sqrt{\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w}}. \tag{4.23}$$

- If $D_+ = (\boldsymbol{\mu}, \boldsymbol{\Sigma})_S$ is a symmetric distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$\begin{cases} b \geq \mathbf{w}^T\boldsymbol{\mu} + \sqrt{\frac{1}{2(1-\gamma)}} \cdot \sqrt{\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w}} & \text{if } \gamma \in (0.5, 1) \\ b \geq \mathbf{w}^T\boldsymbol{\mu} & \text{if } \gamma \in (0, 0.5] \end{cases} \tag{4.24}$$

- If $D_+ = (\boldsymbol{\mu}, \boldsymbol{\Sigma})_{SU}$ is a symmetric and unimodal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$\begin{cases} b \geq \mathbf{w}^T\boldsymbol{\mu} + \frac{2}{3}\sqrt{\frac{1}{2(1-\gamma)}} \cdot \sqrt{\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w}} & \text{if } \gamma \in (0.5, 1) \\ b \geq \mathbf{w}^T\boldsymbol{\mu} & \text{if } \gamma \in (0, 0.5] \end{cases} \tag{4.25}$$

- If $D_+ = G\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ is a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$b \geq \mathbf{w}^T\boldsymbol{\mu} + \Phi^{-1}\left(\gamma\right) \cdot \sqrt{\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w}}, \tag{4.26}$$

  where Φ is the cumulative distribution function.

To sum up, let:

$$\varphi(\gamma) = \begin{cases} \sqrt{\frac{\gamma}{1-\gamma}} & \text{if } \mathbf{x} \sim (\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \sqrt{\frac{1}{2(1-\gamma)}} & \text{if } \mathbf{x} \sim (\boldsymbol{\mu}, \boldsymbol{\Sigma})_S \\ \frac{2}{3}\sqrt{\frac{1}{2(1-\gamma)}} & \text{if } \mathbf{x} \sim (\boldsymbol{\mu}, \boldsymbol{\Sigma})_{SU} \\ \Phi^{-1}(\gamma) & \text{if } \mathbf{x} \sim G(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \end{cases} \tag{4.27}$$

Then for a given γ, the solution b and $\mathbf{w}$ to the problem (4.22) is:

$$-b + \mathbf{w}^T \boldsymbol{\mu}_1 \geq \varphi(\gamma) \cdot \sqrt{\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}}. \tag{4.28}$$

Therefore:

$$\varphi(\gamma) \leq \frac{-b + \mathbf{w}^T \boldsymbol{\mu}_1}{\sqrt{\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}}}. \tag{4.29}$$

Since $\varphi(\gamma)$ is a strictly increasing function of γ, maximizing γ can be obtained by maximizing $\varphi(\gamma)$. The problem (4.21) becomes:

$$\max_{\mathbf{w},b} \quad \frac{-b+\mathbf{w}^T \boldsymbol{\mu}_1}{\sqrt{\mathbf{w}^T \boldsymbol{\Sigma}_1 \mathbf{w}}} \quad \text{s.t.} \quad b \geq \mathbf{w}^T \boldsymbol{\mu}_{-1} \tag{4.30}$$

And the maximum is achieved when $b = \mathbf{w}^T \boldsymbol{\mu}_{-1}$, therefore $\mathbf{w}$ is the solution of:

$$\max_{\mathbf{w}} \frac{\mathbf{w}^T \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)}{\sqrt{\mathbf{w}^T \boldsymbol{\Sigma}_1 \mathbf{w}}}, \tag{4.31}$$

The solution is the eigenvector corresponding to the largest eigenvalue for the following generalized eigenvalue problem :

$$\mathbf{w}^T \boldsymbol{\Sigma}_B \mathbf{w} = \lambda \mathbf{w}^T \boldsymbol{\Sigma}_1 \mathbf{w}, \tag{4.32}$$

where $\boldsymbol{\Sigma}_B = \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)^T$ is the between class scatter matrix. The solution is:

$$\mathbf{w} = \boldsymbol{\Sigma}_1^{-1}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right). \tag{4.33}$$

A direct application of the result is to treat the weak classifier scores as spanning a new feature space, in which we learn a linear classifier with parameter $\mathbf{w}$ and b to optimize the asymmetric learning goal. This post processing stage usually improves the performance. The linear asymmetric classification (LAC) problem (4.31) is very

similar to the Fisher's LDA problem, which solves the following problem:

$$\max_{\mathbf{w}} \frac{\mathbf{w}^T \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right) \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)^T \mathbf{w}}{\mathbf{w}^T \boldsymbol{\Sigma}_W \mathbf{w}}, \tag{4.34}$$

where $\boldsymbol{\Sigma}_W$ is the within-class scatter matrix:

$$\boldsymbol{\Sigma}_W = \sum_{y_i=1} \left(\mathbf{x}_i - \boldsymbol{\mu}_1\right) \left(\mathbf{x}_i - \boldsymbol{\mu}_1\right)^T + \sum_{y_i=-1} \left(\mathbf{x}_i - \boldsymbol{\mu}_{-1}\right) \left(\mathbf{x}_i - \boldsymbol{\mu}_{-1}\right)^T, \tag{4.35}$$

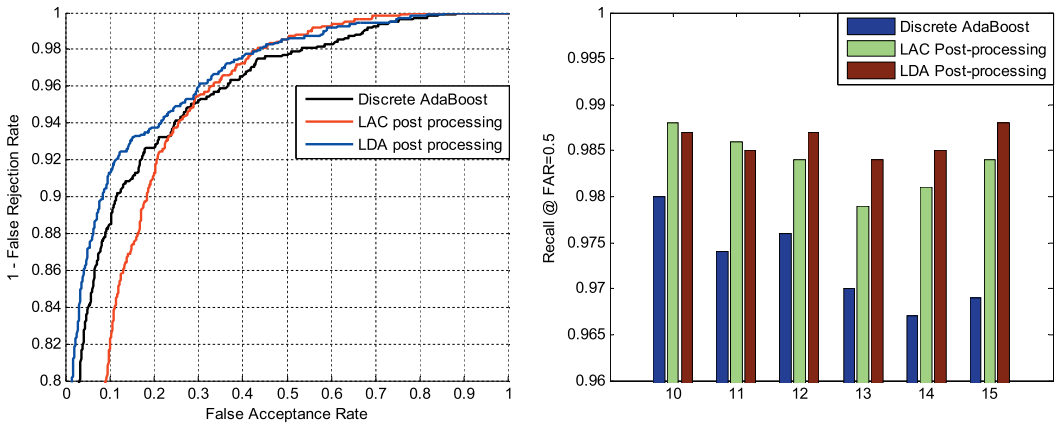
and the optimal solution for Fisher's LDA is $\mathbf{w} = \boldsymbol{\Sigma}_W^{-1} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right)$. Since usually the covariance matrix of the negative class $\boldsymbol{\Sigma}_{-1}$ is near diagonal, therefore the LDA can be treated as LAC with regularization.

An example of applying LAC/LDA post processing to a Discrete AdaBoost classifier is shown in Fig. 4.6a. First we train the Discrete AdaBoost classifier up to selecting 150 weak classifiers, and then the weak classifier weights are re-learned using LAC/LDA on the validation data. The performance of the classifier is evaluated on the standalone test data. We can see both LAC and LDA improves over the original Discrete AdaBoost for $FAR = 0.5$. Furthermore, LAC post processing trades the performance at the low FAR side for improved recall at the medium to high FAR side, while LDA is more balanced, improving the performance almost in the whole range. The performance improvements for stage 10 to 15 in a Boosting cascade detector are shown in Fig. 4.6b. The strong classifier selects 150 weak classifiers, and the recall for $FAR = 0.5$ is shown. We can see the LAC/LDA post processing consistently improves performance, and the improvement is more significant for the later stages, when the training set becomes more difficult.

4.2.5 Boosting with asymmetric goal

The LAC/LDA performs post processing for the weak classifiers' outputs. However, the weak classifiers could be sub-optimal for the asymmetric goal. Though the previously discussed cost-sensitive Boosting can be applied to learn the weak classifiers, but as we have mentioned, there is no explicit relation between the cost-sensitive loss and the ratio of mistakes on the positive and negative set. In this part, we introduce the work in [70], in which a Boosting algorithm is directly derived using the LAC/LDA cost, by applying the forward feature selection technique introduced in section 2.4.

First, we can see that the optimal LAC/LDA solution $\mathbf{w}$ is also the solution of the



(a) (b)

Figure 4.6: Applying LAC/LDA post processing to the weak classifiers of a Discrete AdaBoost classifier. (a) ROC curves. (b) Recall rates at $FAR = 0.5$.

following problem for a proper θ:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w} - \theta\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right). \tag{4.36}$$

θ is the tradeoff parameter corresponding to the maximum eigenvalue of the generalized eigenvalue problem (4.32). The first term measures how concentrated the positive class is, and the second term measures the separation between the positive and the negative classes. Introducing the non-negative and unit L-1 norm constraint on $\mathbf{w}$, we obtain the following problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \tfrac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w} - \theta\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right) \\ \text{s.t.} \quad & \mathbf{w} \geq 0,\ \mathbf{1}^T\mathbf{w} = 1, \end{aligned} \tag{4.37}$$

The Lagrangian formulation of this problem is:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \tfrac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w} - \theta\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right) + \lambda\mathbf{1}^T\mathbf{w}, \\ \text{s.t.} \quad & \mathbf{w} \geq 0. \end{aligned} \tag{4.38}$$

Let a training instance be represented as feature vector $\mathbf{f}_i$ of the weak classifier scores, and let M_1 and M_{-1} represent the number of positive and negative training instances,

respectively, then we can write down the following:

$$\boldsymbol{\mu}_1 = \frac{1}{M_1}\sum_{y_i=1}\mathbf{f}_i,\ \boldsymbol{\mu}_{-1} = \frac{1}{M_{-1}}\sum_{y_i=-1}\mathbf{f}_i,\ \boldsymbol{\Sigma}_1 = \frac{1}{M_1-1}\sum_{y_i=1}\left(\mathbf{f}_i-\boldsymbol{\mu}_1\right)\left(\mathbf{f}_i-\boldsymbol{\mu}_1\right)^T, \tag{4.39}$$

Denote matrix $A^T = [\mathbf{f}_1, ..., \mathbf{f}_M]$, i.e. each row of A is a training instance and each column is a weak classifier/feature, and denote vector $\mathbf{e} = \left[\frac{1}{M_1}, ..., \frac{1}{M_1}, \frac{-1}{M_{-1}}, ..., \frac{-1}{M_{-1}}\right]^T$, then we have:

$$\begin{aligned}\mathbf{w}^T\left(\boldsymbol{\mu}_1-\boldsymbol{\mu}_{-1}\right) &= \mathbf{e}^T A\mathbf{w}\\ \mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w} &= \mathbf{w}^T A^T Q A\mathbf{w},\end{aligned} \tag{4.40}$$

where

$$\begin{aligned}Q &= \begin{bmatrix}Q_1 & 0\\ 0 & 0\end{bmatrix},\\ Q_1 &= \begin{bmatrix}\frac{1}{M} & -\frac{1}{M(M_1-1)} & \cdots & -\frac{1}{M(M_1-1)}\\ -\frac{1}{M(M_1-1)} & \frac{1}{M} & \cdots & -\frac{1}{M(M_1-1)}\\ \vdots & \vdots & \ddots & \vdots\\ -\frac{1}{M(M_1-1)} & -\frac{1}{M(M_1-1)} & \cdots & \frac{1}{M}.\end{bmatrix}\end{aligned} \tag{4.41}$$

To ensure Q is positive definite, we can add a small diagonal term $Q = Q + \sigma I$, and the effect is equivalent to additionally penalizing the sum of squares of the strong classifier scores. Alternatively, we can add σI to $A^T Q A$, which is amount to adding an additional L-2 regularizer of $\mathbf{w}$.

Denote $\rho_i = A_i\mathbf{w}$ and $\boldsymbol{\rho} = A\mathbf{w}$, i.e. ρ_i is the strong learner score of the i-th training instance. Then the problem becomes:

$$\begin{aligned}\min_{\mathbf{w},\boldsymbol{\rho}}\quad & \tfrac{1}{2}\boldsymbol{\rho}^T Q\boldsymbol{\rho} - \theta\mathbf{e}^T\boldsymbol{\rho} + \lambda\mathbf{1}^T\mathbf{w},\\ \text{s.t.}\quad & \mathbf{w}\geq 0,\\ & \rho_i = A_i\mathbf{w},\ i = 1, ..., M.\end{aligned} \tag{4.42}$$

The Lagrangian is:

$$\mathcal{L}\left(\mathbf{w},\boldsymbol{\rho},\boldsymbol{\alpha},\boldsymbol{\beta}\right) = \frac{1}{2}\boldsymbol{\rho}^T Q\boldsymbol{\rho} - \theta\mathbf{e}^T\boldsymbol{\rho} + \sum_{i=1}^{M}\alpha_i\left(\rho_i - A_i\mathbf{w}\right) - \sum_{j=1}^{N}\beta_j w_j + \lambda\mathbf{1}^T\mathbf{w}, \tag{4.43}$$

and the KKT conditions are:

Stationary	Primal and dual feasibility	Complementary slackness
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\rho}} = Q\boldsymbol{\rho} - \theta\mathbf{e} + \boldsymbol{\alpha} = 0$	$\mathbf{w} \geq 0$	$\beta_j w_j = 0$
$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = -A^T\boldsymbol{\alpha} - \boldsymbol{\beta} + \lambda = 0$	$\rho_i = A_i\mathbf{w}$	
	$\boldsymbol{\beta} \geq 0$	

Therefore the new weak classifier selection criterion can be obtained as forward feature selection, by selecting the weak classifier that mostly violates the KKT condition:

$$\arg \max_{j \text{ s.t. } A_{:j}^T\boldsymbol{\alpha} > \lambda} A_{:j}^T\boldsymbol{\alpha}, \tag{4.44}$$

where $A_{:j}$ is the j-th column of A. The dual variables $\{\alpha_j\}$ are obtained from the KKT condition $\alpha_i = \theta e_i - Q_i\boldsymbol{\rho}$, where Q_i is the i-th row of Q:

$$\alpha_i = \begin{cases} \frac{\theta}{M_1} - \frac{M_1}{M(M_1-1)} \left(\rho_i - \frac{1}{M_1} \sum_{j=1}^{M_1} \rho_j \right), & y_i = 1 \\ -\frac{\theta}{M_{-1}}, & y_i = -1 \end{cases} \tag{4.45}$$

Therefore the weak classifiers are encouraged to give negative outputs on all the negative instances regardless of the current strong classifier scores, but the positive instances are downweighted if its strong classifier score ρ_i is higher than the positive mean score, i.e. $\rho_i > \frac{1}{M_1}\sum_{j=1}^{M_1}\rho_j$, and upweighted if ρ_i is below the average, encouraging $\{\rho_i\}$ of the positive instances to be tightly distributed. Furthermore, as done in (3.8), we replace the L-1 norm regularizer by the L-2 norm regularizer, resulting in the following problem:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}_1\mathbf{w} - \theta\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right) + \tfrac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{4.46}$$

This is an unconstrained QP, and its solution can be obtained in closed form as

$$\theta\left(\boldsymbol{\Sigma}_1 + \lambda I\right)^{-1}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right), \tag{4.47}$$

which is essentially identical to solution of the LAC problem (4.33), but adding λ to the diagonals of $\boldsymbol{\Sigma}_1$ for some regularization. However, the weak classifiers are now trained to optimize the LAC criterion, rather than to optimize the AdaBoost loss. Compared with the L-1 regularized problem (4.38), for the unconstrained L-2 regularized problem (4.46), the parameter θ only affects the scale of $\mathbf{w}$. Therefore we only need to test different values of λ for model selection.

Similarly, a Boosting algorithm can be derived for Fisher's LDA cost function. The problem is:

$$\begin{array}{ll} \min\limits_{\mathbf{w}} & \frac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}_B\mathbf{w} - \theta\mathbf{w}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}\right) \\ \text{s.t.} & \mathbf{w} \geq 0,\ \mathbf{1}^T\mathbf{w} = 1. \end{array} \tag{4.48}$$

Σ_B is the between-class scatter matrix. Compared with (4.42), only the matrix Q need to be modifed, the new Q is:

$$\begin{aligned} Q &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix}, \\ Q_2 &= \begin{bmatrix} \frac{1}{M} & -\frac{1}{M(M_{-1}-1)} & \cdots & -\frac{1}{M(M_{-1}-1)} \\ -\frac{1}{M(M_{-1}-1)} & \frac{1}{M} & \cdots & -\frac{1}{M(M_{-1}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{M(M_{-1}-1)} & -\frac{1}{M(M_{-1}-1)} & \cdots & \frac{1}{M} \end{bmatrix}. \end{aligned} \tag{4.49}$$

The performance of Boosting algorithms directly optimizing the LAC cost and the LDA cost are illustrated in Fig. 4.7. We train the stage classifiers for stage 10 to 15 in a Boosting cascade detector, using Discrete AdaBoost, LAC Boost, and LDA Boost. The LAC Boost and LDA Boost uses the same number of weak classifiers as that of Discrete AdaBoost. Compared with the LAC/LDA post processing, the weak classifiers are obtained to optimize the LAC/LDA objective rather than the AdaBoost objective, and the weak classifiers' weights are obtained using the training set rather than the validation set. Comparing Fig. 4.7 and Fig. 4.6, we can observe the additional performance gain due to directly Boosting the LAC/LDA cost.

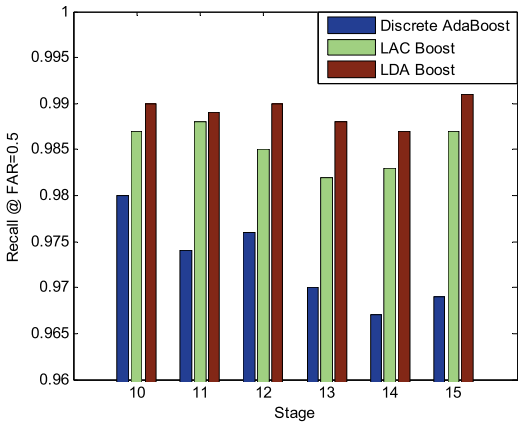


Figure 4.7: Recall rate at $FAR = 0.5$ for Discrete AdaBoost, LACBoost, and LDABoost, for stage 10 $\sim$ 15 in a Boosting cascade detector.

4.3 Operating point selection for detector efficiency

Let stage i of the cascade achieve recall rate a_i and false acceptance rate (FAR) b_i, which are determined by selecting a threshold for the stage classifier's output, i.e. picking an operating point from the ROC curve. A detector with n stages achieves a detection rate $\prod_{i=1}^{n} a_i$ and a false acceptance rate $\prod_{i=1}^{n} b_i$, which should meet the design goal, i.e. $\prod_{i=1}^{n} a_i \geq A$ and $\prod_{i=1}^{n} b_i \leq B$, where A and B are the detector's target detection rate and false positive rate, respectively. Then a designing choice is how to select the threshold and hence the operating points $\{(a_i, b_i)\}$, such that the detector is optimal in terms of efficiency. Since the number of background windows is much higher than the number of objects, the detector's efficiency can be evaluated according to the expected computational cost for rejecting a negative window:

$$\sum_{i=1}^{n} c_i B_{i-1}, \tag{4.50}$$

where c_i is the computational cost of the i-th stage, and $B_i = \prod_{k=1}^{i} b_k$ is the probability that a negative window successfully passes through stage i. Assume $\{b_i\}$ is fixed, then the efficiency can be maximized by minimizing the computational cost $\{c_i\}$ for every stage, while meeting the overall performance requirement (A, B). For the Boosting cascade detector, the complexity of a stage classifier can be measured by the number of selected weak classifiers. For a particular FAR b_i, the recall a_i usually increases with more weak classifiers added, i.e. the ROC curve is pushed upwards by increasing the number of weak classifiers, as illustrated in Fig. 4.8. On the other hand, assume the detector structure is fixed, i.e. the number of weak classifiers for each stage is fixed, the detector efficiency can be optimized by rejecting as many negative windows in the early stages as possible, i.e. adjusting a_i and b_i for each stage as long as the overall Recall/FAR can be achieved.

In this section we introduce the operation point selection method from [83] that optimizes the complexity of a Boosting cascade detector by selecting proper operating points for each stage. The method considers both minimizing the number of weak classifiers and minimizing the false acceptance rate for each stage. The basic assumption of this method is that any point on the ROC curve of the current stage can be reproduced by the ROC curves of later stages, perhaps at the cost of using more com-

plex classifiers. The operating point is selected according to the ROC curve of the validation set.

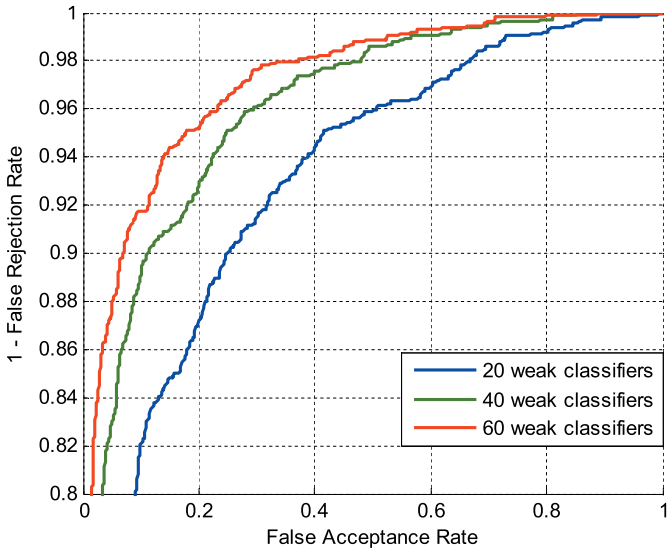


Figure 4.8: The ROC curve of the strong classifier is pushed upwards as more weak classifiers are added.

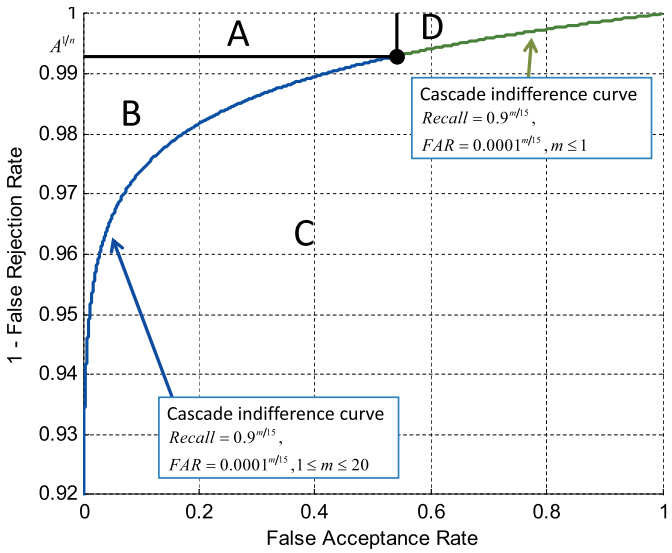


Figure 4.9: The cascade indifference curve and the regions for operating point selection.

4.3.1 Fixed rule

The simplest rule for allocating the $\{(a_i, b_i)\}$ is to set equal values for all stages, i.e.

$$a_i = A^{1/n}, \text{ and } b_i = B^{1/n}. \tag{4.51}$$

Then in each stage we keep adding weak classifiers to the Boosting strong classifier, until the ROC curve reaches or moves above the point (a_i, b_i), i.e. any point of the ROC curve is in region A in Fig. 4.9. A critical limitation of this method is, if the target recall a_i is very high, e.g. $A = 0.90$, $n = 20$ and $a_i = 0.90^{1/20} = 99.5\%$, and if the training set contains a small number of very difficult positive instances, which could be due to labeling noise, then the classifier will desperately add weak classifiers to correctly classify these outliers, though the performance on the validation set may not increase much, i.e. the classifier overfits to the training data and fails to produce the desired generalization performance. On the other hand, if the validation set contains a small number of hard positives, the classifier will never be able to correctly classify these noisy positive instances since it is optimized only for the training set. Therefore the objective (a_i, b_i) may never be reached. To alleviate this problem, and also to improve the efficiency of the detector, the following two methods are proposed in [83], both using more flexible criterion for selecting the operating point.

4.3.2 One-point planning

Rather than using a fixed performance requirement $\left(A^{1/n}, B^{1/n}\right)$ for all stages, the detector performance can be equally achieved by stacking $\leq n$ stages with performance $\left(A^{m/n}, B^{m/n}\right)$, where $1 \leq m \leq n$. The parameterized curve $\{(a^m, b^m)\}$ with m as the controlling parameter is called the cascade indifference curve, as shown in Fig. 4.9. Fixing the value of b, a number of curves can be generated by varying a. Therefore instead of using the fixed rule, we check if the ROC curve is above part of the cascade indifference curve $\left\{\left(A^{m/n}, B^{m/n}\right), 1 \leq m \leq n\right\}$, i.e. any point of the ROC curve is in region A or B in Fig. 4.9. Then by repeating any of such points for n/m times, the detector goal can be reached. It is highly possible that some point in B can be reached before any point in A is reached. For example, the training data contains a small number of very hard positive instances, such that it is difficult to reach a high recall $A^{1/n}$ at a moderate FAR $B^{1/n}$, but it is much easier to reach a lower recall $A^{m/n}$ and a much lower FAR $B^{m/n}$. Therefore fewer weak classifiers are required to reach

$\left(A^{m/n}, B^{m/n}\right)$ than that needed to reach $\left(A^{1/n}, B^{1/n}\right)$, while apparently $\left(A^{m/n}, B^{m/n}\right)$ may also reduce the number of stages needed to reach (A, B), since $m \geq 1$.

If there are multiple points in the region A $\bigcup$ B, we may select the point that has the lowest false acceptance rate (i.e. corresponding to a larger m), to minimize the number of windows passed to the next stage, or conversely, select the point that has highest recall (i.e. corresponding to a smaller m). These two choices correspond to the two intersections of the ROC curve with the boundary of region A and B. While the former choice results in a quite efficient detector by radically rejecting many negative windows in the early stages and possibly using fewer stages, the later is a conservative choice to postpone the rejections to the later stages. We may also select the point on the highest cascade indifference curve to optimize both recall and FAR, i.e. solve for m and A^* in the following problem:

$$\begin{cases} \alpha_k = A^{*m/n} \\ \beta_k = B^{m/n} \end{cases} \Rightarrow \begin{cases} A^* = B^{\log \alpha_k / \log \beta_k} \\ m = n \log \beta_k / \log B \end{cases} \tag{4.52}$$

and accept the point with largest value of A^*. By this choice we may optimize the recall rate for a fixed FAR rate in the final detector, or vice versa.

4.3.3 Two-point planning

It is possible that the ROC curve of the current stage does not contain any point above the cascade indifference curve $\left\{\left(A^{m/n}, B^{m/n}\right), 1 \leq m \leq n\right\}$, i.e. there does not exist any single point on the ROC curve such that the full detector performance (A, B) can be achieved by repeating this point for $\leq n$ stages. Instead, the full detector performance can be achieved by repeating a number of points on the ROC curve for a total of $\leq n$ stages. Denote the set of points on the ROC curve of the current stage as:

$$\left(\alpha_1, \beta_1\right), \cdots, \left(\alpha_K, \beta_K\right). \tag{4.53}$$

Then we want to find a number of points from $\{(\alpha_k, \beta_k)\}$ such that by repeating each point x_k times, and the sum of repetitions does not exceed the required number of stages n, the desired detector performance (A, B) can be achieved, i.e. the optimal

value of the following problem is larger than A:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \prod_{k=1}^{K} \alpha_k^{x_k} \\ \text{s.t.} \quad & \prod_{k=1}^{K} \beta_k^{x_k} \leq B, \ \sum_{k=1}^{K} x_k \leq n, \\ & x_k \geq 0, k = 1, ..., K. \end{aligned} \tag{4.54}$$

By taking logarithm of the objective and the constraints, the problem is equivalent to:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{k=1}^{K} x_k \log \alpha_k \\ \text{s.t.} \quad & \sum_{k=1}^{K} x_k \log \beta_k \leq \log B, \ \sum_{k=1}^{K} x_k \leq n, \\ & x_k \geq 0, k = 1, ..., K. \end{aligned} \tag{4.55}$$

Problem (4.55) is a linear programming problem with K variables and $K+2$ inequality constraints. If the solution exists, the solution must lies on a vertex of the feasible region. Therefore K of the inequality constraints must be tight, and at most two values in $\{x_k\}$ are non-zero. Furthermore, if the optimal value $\geq A$, then the full performance (A, B) can be achieved by repeating these two operating points. Therefore we can stop adding weak classifiers. Among these two points, we may select the one with smaller β_k such that the number of windows passed to the next stage is minimized, and the detector efficiency is maximized. As in the illustration 4.9, the two-point selection method may use point from any region.

The problem (4.54) is formulated such that we achieve the FAR goal while maximizing the recall, and conversely we can formulate a problem to minimize the FAR while achieving the recall as in (4.56). The solution of the two problems are different, and which formulation to use depends on what is more important for the detector, the precision or the recall.

$$\begin{aligned} \min_{\mathbf{x}} \quad & \prod_{k=1}^{K} \beta_k^{x_k} \\ \text{s.t.} \quad & \prod_{k=1}^{K} \alpha_k^{x_k} \geq A, \ \sum_{k=1}^{K} x_k \leq n, \\ & x_k \geq 0, k = 1, ..., K. \end{aligned} \tag{4.56}$$

4.3.4 Non-attainable goal with maximum number of weak classifiers

For all the methods above, a maximum number of allowed weak classifiers should be specified for each stage. If no operating point(s) on the ROC curve satisfies the fixed-rule, one-point or two-point planning goal when the maximum number of weak classifiers are selected, we need an alternative strategy to select a threshold for the stage. The selected threshold must sacrifice the false acceptance rate, the recall rate, or both. For example, if we want to achieve the desired false acceptance rate B, and allow some missed detections, then the alternative strategy for the three methods are as follows:

- For fixed rule, among the points that satisfy the FAR requirement $\left\{\beta_k \leq B^{1/n}\right\}$, select the point with maximum recall α_k.
- For one-point planning, solve (4.52) for m and A^*, and accept the (α_k, β_k) that maximizes A^* while $m \geq 1$, i.e.:

$$\arg\max_k \ \log \alpha_k / \log \beta_k, \quad \text{s.t. } n \log \beta_k / \log B \geq 1. \tag{4.57}$$

- For two-point planning, select the point with smaller β_k even though the optimal value of (4.54) or (4.56) cannot reach the overall requirement.

Similarly, a reversed strategy can be employed such that we achieve the recall rate A, and allow more false positives than B. Then, before selecting the operation point for the next stage, the detector goal for the remaining stages should be updated as follows:

$$A \leftarrow A/a_i \ , B \leftarrow B/a_i \ , \text{and } n \leftarrow n-1. \tag{4.58}$$

Then the operation point of the next stage is selected as if we are learning a cascade with $n-1$ stages, and with the updated performance goal (4.58).

4.3.5 Evaluation

First we show an intuitive example of how the various methods select the operating point in Fig. 4.10a. The classifier is trained as the first stage in a cascade with 15 stages, and the overall objective is $FAR = 0.0001$ and $Recall = 0.9$. The cascade indifference curve and the admissible region for fixed-rule planning are also displayed.

Stage	Fixed Rule		One-Point		Two-Point	
	#WC	Win Ratio	#WC	Win Ratio	#WC	Win Ratio
1	8	1.00	6	1.00	7	1.00
2	12	0.469	11	0.407	8	0.484
3	12	0.254	20	0.207	13	0.251
4	21	0.138	18	0.108	18	8.84×10^{-2}
5	33	7.00×10^{-2}	26	5.88×10^{-2}	23	4.57×10^{-2}
6	40	3.83×10^{-2}	50	2.51×10^{-2}	42	2.57×10^{-2}
7	57	2.08×10^{-2}	50	1.34×10^{-2}	40	9.93×10^{-3}
8	56	1.14×10^{-2}	68	7.18×10^{-3}	69	4.37×10^{-3}
9	95	6.07×10^{-3}	67	3.80×10^{-3}	54	2.18×10^{-3}
10	89	3.37×10^{-3}	110	1.87×10^{-3}	103	1.36×10^{-3}
11	128	1.87×10^{-3}	114	1.07×10^{-3}	200	5.01×10^{-4}
12	89	1.04×10^{-3}	123	6.41×10^{-4}	200	3.81×10^{-4}
13	177	5.69×10^{-4}	118	3.91×10^{-4}	200	2.94×10^{-4}
14	200	3.18×10^{-4}	195	2.47×10^{-4}	200	2.19×10^{-4}
15	200	1.78×10^{-4}	200	1.55×10^{-4}	200	1.56×10^{-4}

Table 4.1: Comparison of the number of weak classifiers and ratio of windows processed by each stage.

The fixed-rule selects from points that satisfy $\alpha_k \geq 0.993$ and $\beta_k \leq 0.541$. The actual selected point by the fixed-rule method is illustrated as a black square, achieving $\alpha_k = 0.994$ and $\beta_k = 0.524$. Using one-point planning, any point on or above the cascade indifferent curve can be selected, and our implemented method select the one with minimum FAR, shown as red dot in the figure, with $\alpha_k = 0.990$ and $\beta_k = 0.428$. Using the two-point planning method that optimizes problem (4.54), the selected point is shown as the green dot with purple boundary, which is below the cascade indifference curve, with $\alpha_k = 0.0.989$ and $\beta_k = 0.0.371$. The two-point planning problem (4.54) selects two points in the solution, and the other point is shown as the green dot on the top right. We can see that the one-point and two-point planning improves the detector efficiency by rejecting more negative instances than the fixed-rule method, at the expense of rejecting more true positives. The full detector performance is guaranteed by the assumption of repeatability, i.e. any point on the current ROC curve can be reproduced by ROC curves of later stages. In addition, the one-point and two-point planning may further improve the detector efficiency by using fewer weak classifiers, as they make use of a larger set of points from the ROC curve.

The number of weak classifiers and portion of windows processed by each stage are listed in Table 4.1 for the fixed rule, one-point planning, and two-point planning methods. Each stage is limited to use a maximum of 200 weak classifiers. The recall

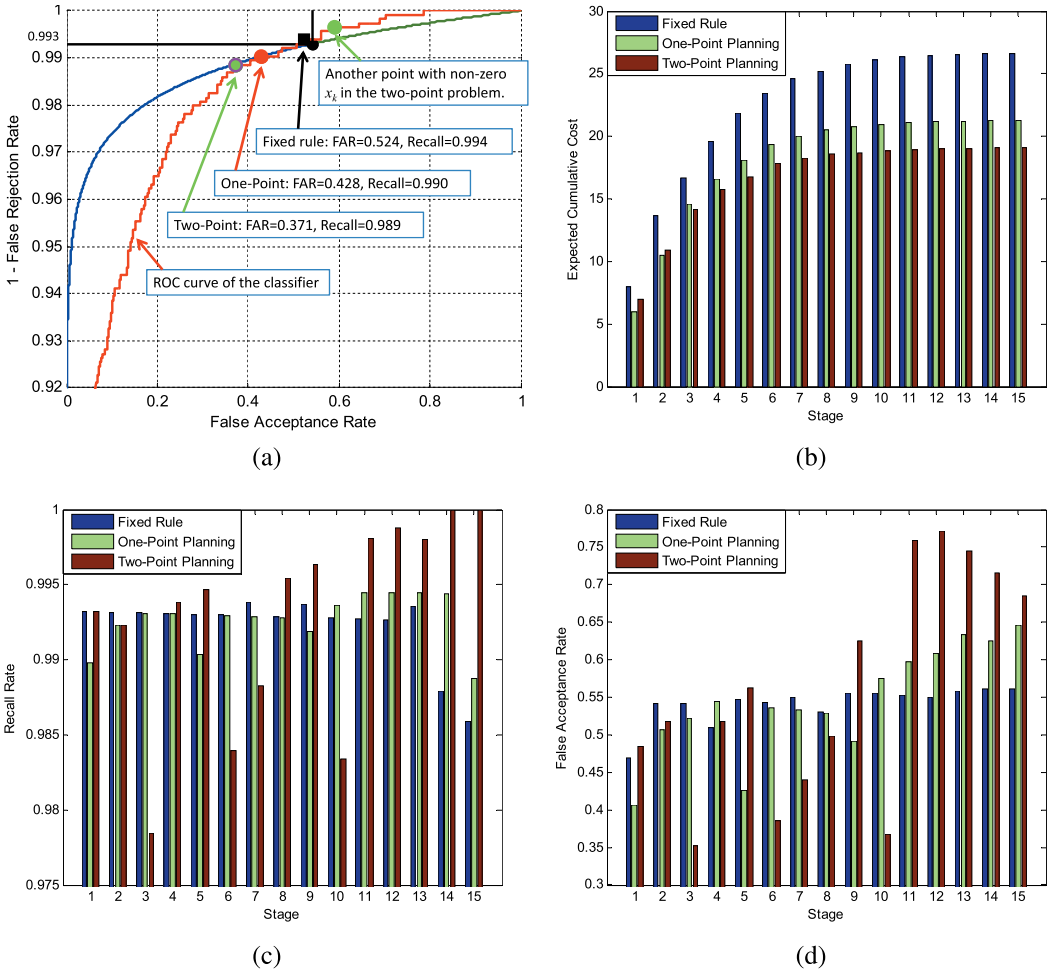


Figure 4.10: (a) An example of the three operating point selection methods. (b) Expected cumulative costs (number of weak classifiers) of evaluating a window for each planning method. (c) The recall rates. (d) The False Acceptance Rates.

rate and false acceptance rate of each stage are shown in Fig. 4.10c and 4.10d. The expected cost up to each stage is shown in Fig. 4.10b. We can see both one-point planning and two-point planning produces more efficient detectors than the fixed rule method. The benefit comes from both the smaller number of weak classifiers used, and the quick elimination of negative windows in the early stages. We also find that the fixed rule method fails to reach the Recall/FAR objective for the last two stages, reducing the overall recall by about 1%, while the violation is much smaller for the one-point planning and two-point planning, only reducing the overall recall by 0.5% and 0.3%, respectively. Furthermore, it can be observed that the two-point planning method reduces the recall very much in the early stages, such that the later stages need very high recall rate in order to achieve the overall recall. Therefore the last few

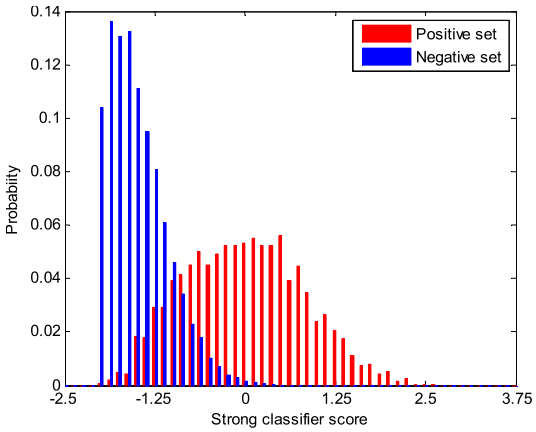


Figure 4.11: Distribution of the scores of the training set for stage 10 of a cascade detector using the classifier of stage 9.

stages of the cascade reach the limit of maximum weak classifier number, and also have relatively high false acceptance rates. But since only a very small portion of windows ($\sim 0.05\%$) need to be processed by the later stages, the detector's efficiency is not impaired.

4.4 Information recycling in cascade

In a cascade detector, it is usually believed that the earlier stages of a cascade contain valuable information for the classification task of the later stages. For example, as shown in Fig. 4.11, it can be clearly observed that the negative training instances for the new stage, which are collected from false positives of the previous stage, also have a much lower average score than the positive instances, when evaluated by the classifier of the previous stage. The reason is that the operation point of the previous stage is chosen for a high detection rate (e.g. 0.99) and a moderate false positive rate (e.g. 0.5). The classification task of the current stage could be much easier if the information from the previous stages can be re-used. For the Boosting cascade detector, recycling the information may lead to fewer new weak classifiers or fewer new features in the new stage, thus improving the detector efficiency.

Let $F_t(x) = \sum_j w_{t,j} f_{t,j}(x)$ be the strong classifier for the t-th stage, and each weak classifier $f_{t,j}(x)$ uses a feature $x_{t,j}$. The information from the previous cascade stage can be exploited in several ways:

Strong classifier chaining We may treat the previous stage strong classifier score as the starting point of the strong classifier, and continue training by adding more weak classifiers to the Boosting strong classifier as in [72, 73, 74, 75]. Then the strong classifier for the new stage is:

$$F_{t+1}(x) = F_t(x) + \sum_j w_{t+1,j} f_{t+1,j}(x). \tag{4.59}$$

A problem with this approach is, for many training instances, the previous strong classifier already incur a large margin $yF_t(x)$. Therefore in the AdaBoost scheme, their weights $\exp(-yF_t(x))$ will be very small, and the training process is dominated by a few very hard instances, which could impair the generalization performance. In this sense, this strategy entirely inherits the bias of the training set of the previous stage.

Strong classifier nesting The second method is to treat $F_t(x)$ as a feature, and train the first weak classifier using this feature, as in [111]. The strong classifier for the new stage is:

$$F_{t+1}(x) = f(F_t(x)) + \sum_j w_{t+1,j} f_{t+1,j}(x). \tag{4.60}$$

Compared with strong classifier chaining, less bias is introduced in the first weak classifier. However, the information from the previous stages has not been fully exploited, since the weak classifier is usually limited to a simple form, e.g. a decision stump.

Weak classifier recycling by totally-corrective Boosting The totally-corrective Boosting algorithm can be applied to reuse the weak classifiers, e.g. in [112]. The previous weak classifiers are considered to be already included in the strong classifier of the new stage, but their weights are decided by solving the totally-corrective Boosting problem on the new dataset. Then new weak classifiers are introduced for the new stage in the usual manner. The strong classifier for the new stage classifier has the following form:

$$F_{t+1}(x) = \sum_{j=1}^{N_t+N_{new}} w_{t+1,j} f_{t+1,j}(x), \tag{4.61}$$

where the new strong classifier contains all the N_t weak classifiers from the previous stages, and the N_{new} weak classifiers added in the new stage. If the L-1 regularized totally-corrective Boosting is employed, the solution will select only a subset of the N_t old weak classifiers, while the rest are assigned with weight 0, indicating that they are less relevant to the new problem. By assigning a new set of weights to the previous

weak classifiers, the bias due to the previous training set can be effectively reduced.

Feature recycling We can consider re-using the features $\{x_{t,j}\}$ extracted by the previous strong classifier, while discarding all other information, i.e. the weak classifiers' weights $\{w_{t,j}\}$ and the parameter of the weak classifiers $\{f_{t,j}(x)\}$ (e.g. threshold and polarity for a decision stump). This method introduces least bias from the previous training dataset, but also incurs the most overhead in exploiting the existing information. However, to the best of our knowledge, no work has been performed for feature recycling in the literature.

In this section, we introduce a biased selection strategy for recycling the information from the previous stages of the Boosting cascade detector. The strategy can be applied for recycling both the weak classifiers and the features. Only those weak classifiers/features helpful for the new classification problem are re-used, while those less relevant ones will not be included in the new stage.

4.4.1 Biased selection strategy for weak classifier recycling

We first introduce a biased weak classifier selection strategy for recycling the weak classifiers, using the stage-wise additive regression formulation rather than the totally-corrective formulation. As shown in the previous sections, selecting a fixed number of weak classifiers, the totally-corrective formulation achieves lower risk on the training data than the non-totally-corrective way. But if sufficiently many weak classifiers are selected, the generalization performance is similar between the two methods.

Let $\mathcal{F}_{recycle} = \{f\}_{N_{recycle}}$ be the set of weak classifiers already included in the detector up to the current point, and $\mathcal{F}_{all} = \{f\}_{N_{all}}$ be the set of all the weak classifiers. Then $\mathcal{F}_{recycle}$ is the set of weak classifiers that can be recycled, and $\mathcal{F}_{other} = \mathcal{F}_{all} - \mathcal{F}_{recycle}$ is the set of weak classifiers not included in the current detector, i.e. those that if selected as the next weak classifier would increase the complexity of the detector. Note that each feature and each parameterization of the weak classifier is counted as a unique weak classifier. For example, obviously two decision stumps using different features are two different weak classifiers. Furthermore, two decision stumps using the same feature, but with different thresholds are also counted as two different weak classifiers. Then in each Boosting round, we need to decide if we should recycle an existing weak classifier in $\mathcal{F}_{recycle}$, or introduce a new one from $\mathcal{F}_{other}$. For this purpose, we find the best weak classifier $f_{recycle,best}$ in $\mathcal{F}_{recycle}$ and the

best weak classifier $f_{other,best}$ in $\mathcal{F}_{other}$, and see how much improvement can be made by including either as the selected weak classifier for this Boosting round. For Discrete AdaBoost, we calculate the weight $w_{recycle,best}$ and $w_{other,best}$ for the new weak classifier, and evaluate the new empirical exponential loss. Denote the reduction in the empirical loss as $d_{recycle,best}$ and $d_{other,best}$, respectively. If we find that $d_{other,best}$ is not signifcantly better than $d_{recycle,best}$, e.g. $d_{other,best} - d_{recycle,best} \leq \lambda$, we decide that the additional complexity introduced by a new weak classifier is not worthy, and recycling an existing one may be a better choice. Otherwise, the best new weak classifier from $\mathcal{F}_{other}$ is used. The algorithm is summarized in Algorithm 3.

Algorithm 3 Train Boosting cascade stage with weak classifier recycling

Initialization: $\mathcal{F}_{recycle}$ and $\mathcal{F}_{other}$, the strong classifier $F = 0$.
for $k = 1$ to K **do**
  Find the best weak classifier $f_{recycle,best}$ and $f_{other,best}$ from $\mathcal{F}_{recycle}$ and $\mathcal{F}_{other}$, respectively.
  Decide the weights for $f_{recycle,best}$ and $f_{other,best}$, and measure the improvement in the loss function:

$$d_{other,best} = L(F) - L(F + w_{other,best} f_{other,best})$$
$$d_{recycle,best} = L(F) - L(F + w_{recycle,best} f_{recycle,best})$$

  if $d_{other,best} - d_{recycle,best} > \lambda$ **then**
    Select the new weak classifier $f_{other,best}$ into the strong classifier:

$$F(\mathbf{x}) \leftarrow F(\mathbf{x}) + w_{other,best} f_{other,best}(\mathbf{x})$$
$$\mathcal{F}_{recycle} \leftarrow \mathcal{F}_{recycle} \bigcup \{f_{other,best}\}$$
$$\mathcal{F}_{other} \leftarrow \mathcal{F}_{other} \backslash \{f_{other,best}\}$$

  else
    Update the weight of the existing weak classifier $f_{recycle,best}$ by $w_{recycle,best}$:

$$F(\mathbf{x}) \leftarrow F(\mathbf{x}) + w_{recycle,best} f_{recycle,best}(\mathbf{x}).$$

  end if
end for
return $\mathcal{F}_{recycle}$, $\mathcal{F}_{other}$, and F.

If a fixed threshold λ is used in Algorithm 3, the algorithm can be interpreted as an approximate solution of the following problem:

$$\min \quad J(F, \mathcal{F}_{recycle}) = L(F) + \lambda Card(\mathcal{F}_{recycle}), \tag{4.62}$$

where J is the regularized loss function, composed by the empirical loss L, and penalty on the number of unique weak classifiers used in the detector. F is the strong classifier of the current stage, $\mathcal{F}_{recycle}$ is the set of weak classifiers used in the detector, including both weak classifiers used the previous stages and weak classifiers that only appear in the new stage. $Card(\mathcal{F}_{recycle})$ counts the number of unique weak classifiers used in

the detector. Algorithm 3 solves this problem by adding one weak classifier each time. A new weak classifier is added if only it provides sufficient advantage over adjusting the weight of a existing weak classifier, i.e. $d_{other,best} - d_{recycle,best} > \lambda$.

Notice that since we do the update $\mathcal{F}_{recycle} \leftarrow \mathcal{F}_{recycle} \bigcup \{f_{best}\}$ when a new weak classifier is added, the algorithm not only encourages reusing weak classifiers in the previous stages, but also encourages reusing weak classifiers selected in the earlier Boosting iterations of the current stage. Furthermore, we treat weak classifiers whose output values differ only by a constant value or a constant factor as the same weak classifier. For example, if two decision stumps uses the same feature and threshold, but differ in polarity, or two regression stumps differ only in the output value, they will be treated as one weak classifier. Therefore Algorithm 3 allows decreasing of weak classifier weights. The effect would be similar to the totally-corrective Boosting, in which all the selected weak classifiers' weights are adjusted in each iteration. By contrast, our algorithm adjust only one weak classifiers' weight in each iteration, by all the selected weak classifiers' weights can be adjusted in each iteration. Our algorithm is less radical in minimizing the empirical loss, such that new weak classifiers can be introduced before the weights of the existing weak classifiers are adjusted to the optimum of the totally-corrective Boosting problem (3.7).

In Algorithm 3, the parameter λ controls the behavior of feature recycling. Large value for λ encourages feature recycling, since a new weak classifier will be selected only when it has a significant competitive edge over an existing weak classifier. Conversely, small value for λ emphasizes reducing the loss $L(F)$. For AdaBoost training, the exponential loss function is used, and the empirical loss converges in exponential speed. Therefore the room for improvement decreases with more weak classifiers selected. If a fixed λ is used, no new weak classifiers will be introduced in the later Boosting iterations, since $\max(d_{other,best}, d_{recycle,best}) \leq \lambda$. To deal with this issue, we use an adaptive λ during the training process. In particular, we set $\lambda = \lambda^* \max(d_{other,best}, d_{recycle,best})$. The parameter λ^* controls the introduction of new weak classifiers in a similar way as the parameter λ in the totally-corrective Boosting problem (3.7). However, working in a stage-wise additive manner, we may control λ^* more freely than the previous totally-corrective Boosting method.

The proposed weak classifier recycling scheme has a potential drawback in that the sum of weak classifier weights may goes unbounded since it lacks any mechanism for constraining the weak classifier weights. Therefore it is possible that Algorithm 3 has many recycling steps, and the sum of weak classifier weights will be large while only a small number of unique weak classifiers are used, which may causes overfit to the

training data. By contrast, the ordinary AdaBoost constrains the sum of weak classifier weights by selecting a finite number of weak classifiers, and the totally-corrective AdaBoost explicitly introduces the L-1/L-2 regularization. We introduce two approaches to alleviate this problem. First, with the stage-wise additive scheme which is not totally-corrective, we apply a shrinking procedure if a weak classifier already used in the *current strong classifier* is recycled. After the weak classifier is recycled and its weight in the strong classifier is adjusted, we normalize the weak classifier weights $\mathbf{w}$ such that its L-1 norm is not changed before and after the recycling step. Second, we solve the totally-corrective Boosting with the selected the weak classifiers. Notice that when using the second approach for recycling weak classifiers from previous stages of the cascade, the recycled weak classifiers are introduced one by one, using the biased selection strategy, while the existing totally-corrective Boosting approach for weak classifier recycling introduces all the weak classifiers in the beginning, and relies on the sparsity-inducing L-1 norm regularization to decide which are relevant and should be have non-zero weights.

4.4.2 Biased selection strategy for feature recycling

A similar method can be applied for recycling features instead of weak classifiers, such that we reuse the previously extracted features, but the weak classifiers are adjusted to better fit the new training data.

Let $\mathcal{X}_{recycle} = \{x\}_{N_{recycle}}$ be the set of features included in the detector up to the current stage. Let $\mathcal{F}_{recycle} = \{f(x)\}_{x \in \mathcal{X}_{recycle}}$ be the set of weak classifiers only using features from $\mathcal{X}_{recycle}$. We also denote $\mathcal{F}_{all} = \{f\}_{N_{all}}$ be the set of all weak classifiers, and $\mathcal{F}_{other} = \mathcal{F}_{all} - \mathcal{F}_{recycle}$ be the set of weak classifier not using existing features.

The algorithm is similar to Algorithm 3, such that we select $f_{other,best}$ if it has a significant advantage over selecting $f_{recycle,best}$, decided by the following criterion:

$$d_{other,best} - d_{recycle,best} > \lambda.$$

For a fixed threshold λ, the algorithm approximately solves the following problem:

$$\min \quad J(F, \mathcal{X}_{recycle}) = L(F) + \lambda Card(\mathcal{X}_{recycle}) \tag{4.63}$$

If we consider weak classifiers using a particular feature as a group of weak classifiers, the regularization term $Card(\mathcal{X}_{recycle})$ penalizes for the number of groups used in the detector. Therefore the algorithm does not penalize for multiple weak classifiers

using the same feature. Essentially multiple weak classifiers using a particular feature can be interpreted as a more weak classifier, e.g. several decision stumps together can be taken as a decision tree, as illustrated in Fig. 4.12.

4.4.3 Experimental Evaluation

First, we demonstrate the biased selection recycling scheme, but without any a priori weak classifiers from the earlier stages. Therefore the Boosting process tries to recycle weak classifiers selected in the earlier Boosting iterations. The ordinary AdaBoost, totally-corrective AdaBoost, and AdaBoost with weak classifier recycling are compared. The decision stumps are used as weak classifiers. For the totally-corrective AdaBoost, the parameter λ is selected by cross-validation, and the optimal value is $\lambda = 0.1$. For the biased selection strategy for recycling weak classifiers, shrinking of

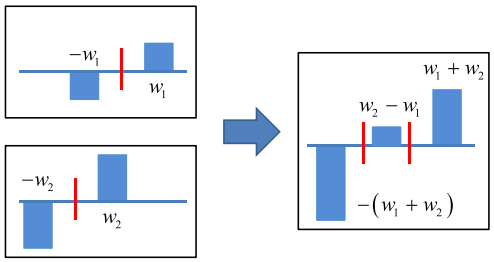
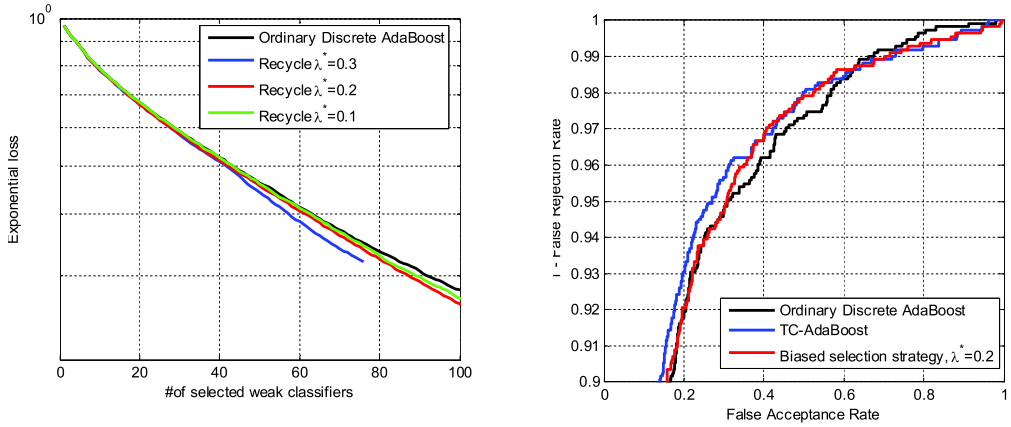


Figure 4.12: Reusing a feature in multiple weak classifiers is equivalent to using a complex weak classifiers with this feature.



(a) (b)

Figure 4.13: (a) Convergence of exponential loss using the weak classifier recycling methods. (b) Comparison of ROC curves of ordinary AdaBoost, totally-corrective AdaBoost, and ordinary AdaBoost with weak classifier recycling.

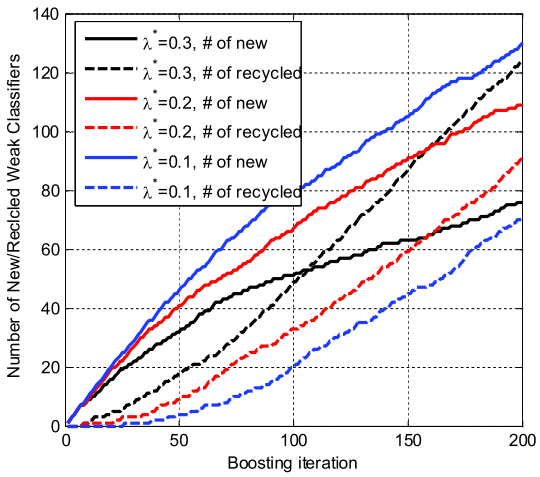


Figure 4.14: Number of new weak classifiers and recycled weak classifiers during the Boosting learning process for various λ.

weak classifier weights is performed, and we show the results for $\lambda^* = 0.1, 0.2, 0.3$. The convergence of exponential loss for ordinary AdaBoost and recycling with the biased selection strategy are shown in Fig. 4.13a. It is observe that a large λ^* leads to a lower value of the exponential loss, since the weak-classifier recycling procedure fine-tunes the weak classifier weights. On the other hand, a small λ^* leads to a similar performance to that of the ordinary AdaBoost, indicating that weak-classifier recycling happens more rarely. Also notice that we train the strong classifier for at most 200 Boosting iterations, and less than 80 unique weak classifiers are obtained for $\lambda^* = 0.3$, since many Boosting iterations are spent in refining the weights of existing weak classifiers. The ROC curves on the test data are shown in Fig. 4.13b for using 50 weak classifiers. For the biased selection strategy, we show the result for $\lambda^* = 0.2$, since 0.3 seems to overfit on the training data. We can see the step-wise recycling results in a significantly improvement over the ordinary AdaBoost, and is comparable to the TC-AdaBoost. Notice that the TC-AdaBoost solves for the optimal weak classifiers' weights exactly. The number of new weak classifiers and recycled weak classifiers are shown in Fig. 4.14 with respect to the number of Boosting iterations. Notice that, when using weak-classifier recycling, we need more than N Boosting iterations for selecting N unique weak classifiers, since some iterations will adjust the weight of an existing weak classifier rather than introducing a new one. We can see that most recycling steps happen in the later Boosting iterations, implying that a number of diverse weak classifiers will be selected in the early iterations, and then their weights are refined in the later iterations.

We also test the biased selection scheme for feature recycling, which still recycles only features selected in the earlier Boosting iterations. Decision stumps are used as the weak classifiers, and the ROC curves of using 50 features are shown in Fig. 4.15, for $\lambda^* = 0.1, 0.2, 0.4, 0.6$. The actual numbers of weak classifiers used are 58, 95, 189, and 344, respectively. All settings significantly improve over those that do not use feature recycling. The distribution of the frequency that a feature is used is shown in Fig. 4.16. As expected, large λ^* causes more features to be recycled more often. Since several stumps using a particular feature can be combined as a tree, the feature recycling essentially allows the use of more complex weak classifiers (e.g. multi-node trees rather than stumps) for those informative features, while using simple weak classifiers for those less informative features, rather than fixing the weak-classifier structure for all features.

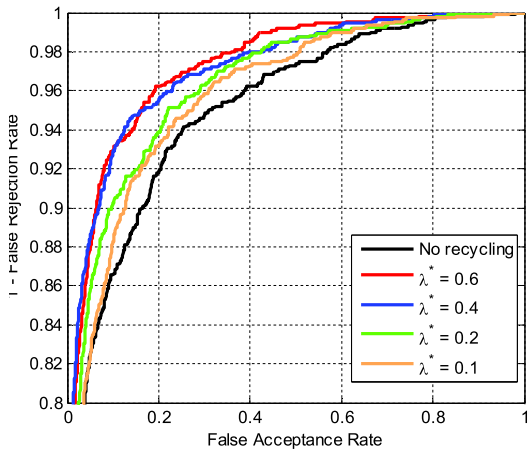


Figure 4.15: ROC curves of the Boosting classifier trained with feature recycling, selecting up to 50 features.

To compare the performance of various information-recycling cascades, we train a fully functional Boosting cascade detector with information recycling. We compare between three different methods. First, the strong classifiers' scores are directly-reused (SL-Rec), treating them as the first weak classifier in the next stage. Second, all existing weak classifiers are introduced in training the new stage in the beginning, using L-1 regularized totally-corrective AdaBoost. We call this method Batch-Rec since it recycles all weak classifiers in a batch operation. Third, the biased selection strategy is employed such that an existing weak classifier $f_{recycle,best}$ is recycled only when it is better than the best new weak classifier $f_{other,best}$, and then we solve a L-2 regularized totally-corrective AdaBoost to decide the weak classifiers' weights. This method is

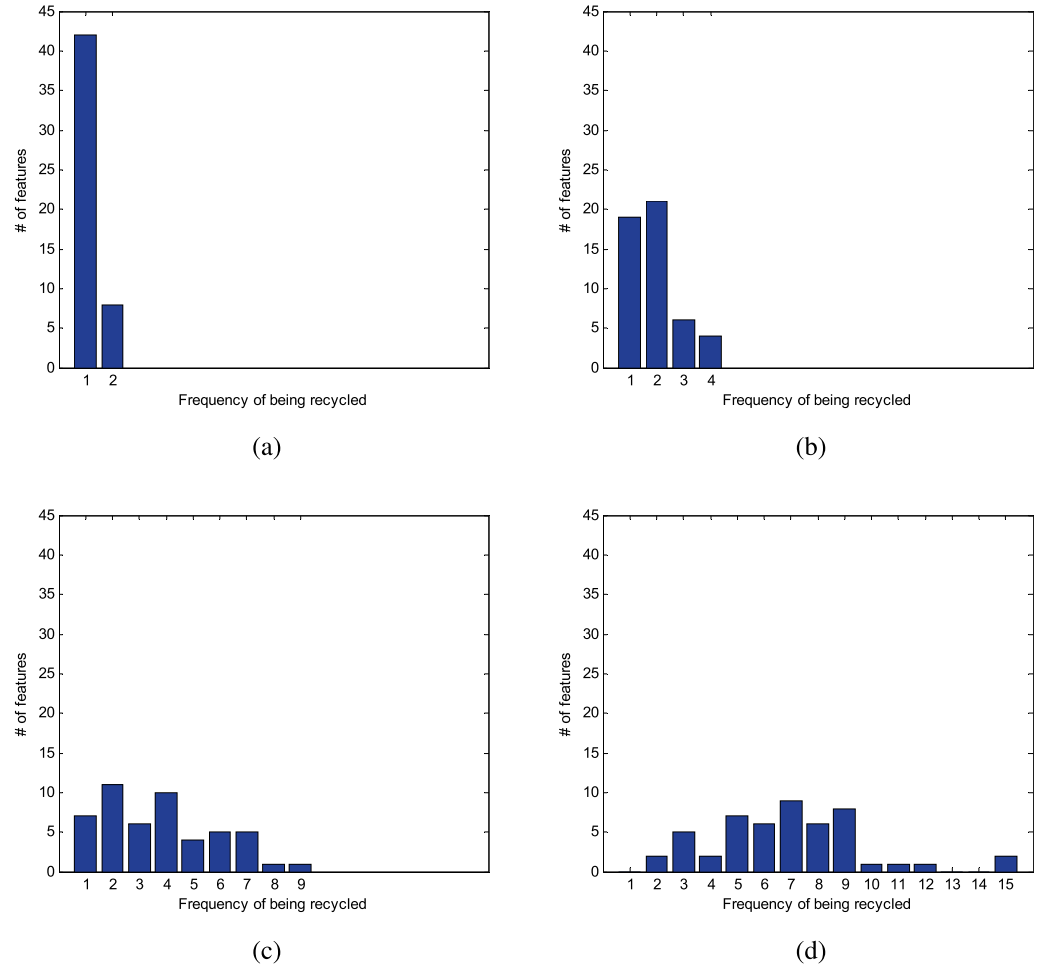


Figure 4.16: Frequency of feature recycling for selecting up to 50 features for (a) $\lambda^* = 0.1$; (b) $\lambda^* = 0.2$; (c) $\lambda^* = 0.4$; (d) $\lambda^* = 0.6$.

referred to as BS-Rec in the following. Notice that, in comparison with L-1 regularization, L-2 regularization does not result in sparse solutions. Instead, we selectively recycle a subset of the existing weak classifiers by the step-wise biased selection criterion. An analysis of the resulting detectors is shown in Fig. 4.17. The number of new weak classifiers introduced in each stage is shown in Fig. 4.17a. We can see the two weak-classifier recycling methods result in much fewer new weak classifiers than the strong-classifier recycling method does, especially in the later stages, which is due to the face that the weak classifiers' weights are not adapted to the new training data for the SL-Rec method. The step-wise approach (BS-Rec) further reduces the number of new weak classifiers as compared to Batch-Rec. Furthermore, as shown in Fig. 4.17c, BS-Rec recycles fewer weak classifiers than Batch-Rec. Both results indicate

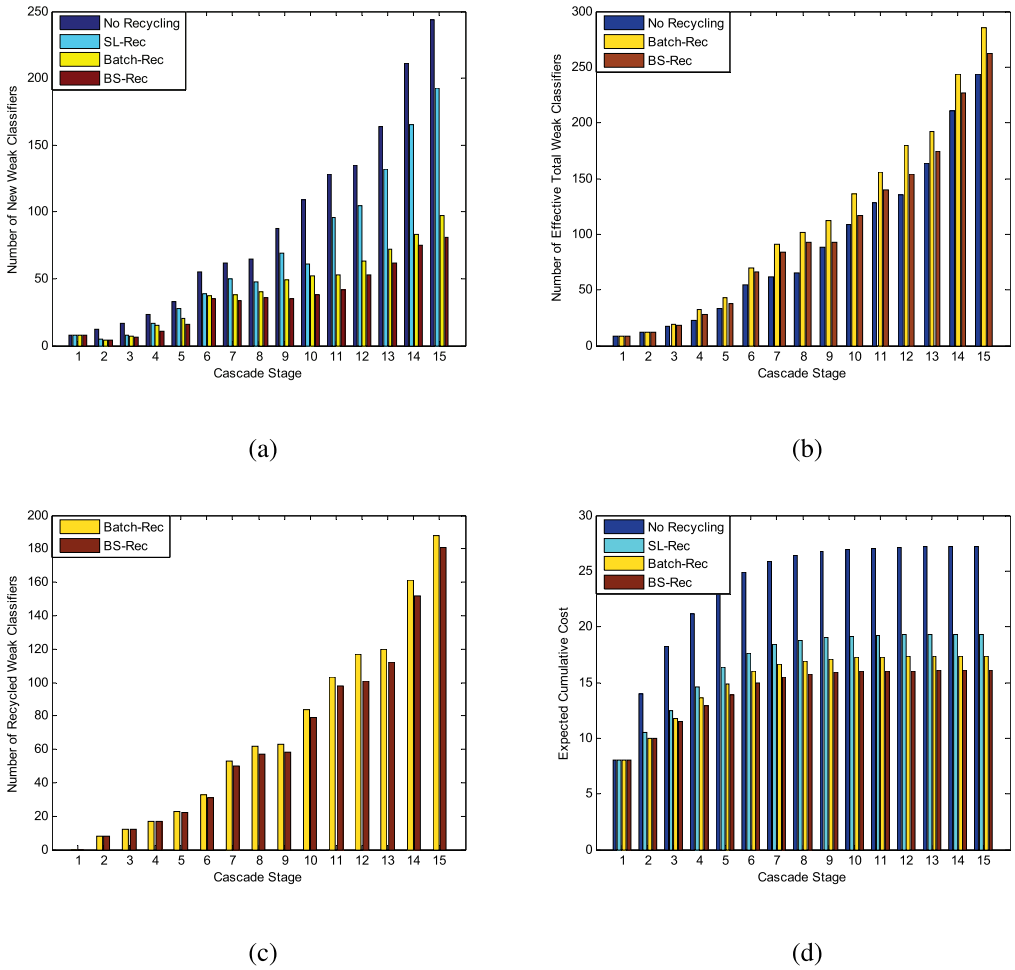


Figure 4.17: Train a cascade detector with information recycling. (a) Number of new weak classifiers added in each stage; (b) Number of effective weak classifiers used in each stage; (c) Number of recycled weak classifiers in each stage; (d) Expected cost of evaluating the detector up to each stage.

that it can be counter-productive to include all existing weak classifiers in the beginning, while the step-wise approach alleviates this problem. The number of effective weak classifiers, i.e. both new weak classifiers and recycled weak classifiers, for each stage are shown in Fig. 4.17b. As expected, no recycling results in the least number of effective weak classifiers, since $f_{recycle,best}$ are almost always inferior to $f_{other,best}$ in terms of reduction of the empirical loss $L(F)$. Consequently, we need more recycled weak classifiers to achieve the same performance level than using only new trained weak classifiers. The detector efficiency for various methods, measured by the expected number of weak classifiers evaluated for a window during sliding window detection, is shown in Fig. 4.17d. From the results, we can observe a clear efficiency

improvement using the BS-Rec method.

4.4.4 Conclusions and future works

We have proposed a method for effectively exploiting the available information in training a Boosting cascade visual object detector. By biasing our preference of weak classifiers towards those that reuses available information, e.g. the weak classifiers that are already included in the detector and the features that are already extracted, our method strikes a balance between the complexity of the detector and the reduction of training objective function. Our method assigns a new set of weights for those recycled weak classifiers, as done by the totally-corrective Boosting method. But rather than directly incorporating all previous weak classifiers into the problem, our method reuses an existing weak classifier only when it is advantageous, and better performance is observed in practice. The proposed biased selection strategy naturally extends to recycling features, and provides an adaptive control of the complexity of weak classifiers, such that for a highly informative feature, a powerful weak classifier can be obtained by merging several ordinary weak classifiers. We believe our method is a meaningful extension to the currently popular Boosting cascade visual object detectors. Further research would involve better recycling criterion, and a in-depth analysis of optimizing the detector efficiency.

Chapter 5 Kernel methods and the similarity features

Kernel methods are powerful techniques for solving difficult learning problems, for example, classification with non-linear decision boundaries. By mapping the original features to an implicit feature space Φ in which the inner product is defined via a kernel function k, linear classifiers in Φ can capture non-linear separating boundaries in the original feature space. In this chapter we discuss the following issues for kernel methods. In section 5.1, we give a brief introduction of the rationale behind the kernel trick, along with two kernel methods, i.e. kernel SVM and kernel PCA. Methods for improving the efficiency of the kernel SVM are discussed in section 5.2. The efficiency of evaluating a classifier is essential for time-demanding applications such as visual object detection. Lastly, inspired by the kernels as similarity metrics, we introduce the multi-kernel, multi-instance similarity feature for visual object detection in section 5.3.

5.1 The kernel trick

The kernel trick assumes there is a feature mapping $\phi(\mathbf{x}) : \mathcal{X} \rightarrow \Phi$, and a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, such that the inner product in the feature space Φ is equal to the kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \tag{5.1}$$

We are going to learn a linear model in the feature space Φ: $f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$. By some manipulation, only the inner product of the training instances $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ appears in the learning problem, and the solution is in the subspace spanned by the training instances, i.e. $\mathbf{w} = \sum_{i=1}^{M} \alpha_i \phi(\mathbf{x}_i)$. The linear model becomes:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{5.2}$$

The set of instances $\{\mathbf{x}_1, ..., \mathbf{x}_M\}$ used in the model are called the basis vectors through-

out this chapter. Then only the kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ needs to be provided, while the feature mapping $\phi(\mathbf{x})$ does not need to be explicitly known. Furthermore, $\phi(\mathbf{x})$ can be a very high dimensional vector, such that directly learning with the feature space Φ is impractical. Then the learning problem solves for the parameters $\{\alpha_i\}$ instead of $\mathbf{w}$.

5.1.1 Kernel SVM

Dropping the bias term in the linear scoring function, the SVM learning problem in the feature space Φ is:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & y_i\mathbf{w}^T\phi(\mathbf{x}_i) \geq 1 - \xi_i,\ i = 1, ..., M, \\ & \xi_i \geq 0,\ i = 1, ..., M. \end{aligned} \tag{5.3}$$

The dual problem can be written as:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\boldsymbol{\alpha}^T K \boldsymbol{\alpha} - \sum_{i=1}^{M} y_i\alpha_i \\ \text{s.t.} \quad & 0 \leq y_i\alpha_i \leq C, i = 1, ..., M. \end{aligned} \tag{5.4}$$

K is a $M \times M$ matrix, and $K_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_i) = k(\mathbf{x}_i, \mathbf{x}_j)$. The primal solution $\mathbf{w}$ can be obtained from the dual problem (5.4) as $\mathbf{w} = \sum_{i=1}^{M}\alpha_i\phi(\mathbf{x}_i)$, and the scoring function can be represented as:

$$f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{i=1}^{M}\alpha_i\phi(\mathbf{x}_i)^T\phi(\mathbf{x}) = \sum_{i=1}^{M}\alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{5.5}$$

Hence the SVM is kernelized, such that only $k(\mathbf{x}_i, \mathbf{x}_j)$ need to be provided, and the dual problem is solved to obtain $\boldsymbol{\alpha}$. Furthermore, as shown in section 2.2, the dual solution $\boldsymbol{\alpha}$ is a sparse vector such that only those on-margin and in-margin training instances, i.e. $\left\{i : y_i\mathbf{w}^T\mathbf{x}_i \leq 1\right\}$, will have $\alpha_i \neq 0$, which are called the support vectors, and $f(\mathbf{x})$ is fully decided by the support vectors.

5.1.2 Kernel PCA

Principal component analysis (PCA) finds an n-d subspace of the original feature space with orthonormal basis $B = [\mathbf{b}_1, ..., \mathbf{b}_n]$, such that the projection in this subspace is the best approximation of the original data:

$$\min_B \sum_{i=1}^{M} \left\| \mathbf{x}_i - BB^T\mathbf{x}_i - \mathbf{c} \right\|^2 \quad \text{s.t.} \quad B^TB = I. \tag{5.6}$$

Taking the derivative of the objective function with respect to $\mathbf{c}$, and setting the derivative to 0, we can obtain:

$$\mathbf{c} = \frac{1}{M}\left(\sum_{i=1}^{M} \mathbf{x}_i - BB^T \sum_{i=1}^{M} \mathbf{x}_i \right). \tag{5.7}$$

Substituting (5.7) into (5.6), then B can be obtained by solving the following problem:

$$\min_B \sum_{i=1}^{M} \left\| \hat{\mathbf{x}}_i - BB^T\hat{\mathbf{x}}_i \right\|^2, \quad \text{s.t.} \quad B^TB = I, \tag{5.8}$$

where $\hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{M}\sum_{i=1}^{M} \mathbf{x}_i$. The problem is equivalent to:

$$\max_B \sum_{j=1}^{n} \mathbf{b}_j^T \left(\sum_{i=1}^{M} \hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T \right) \mathbf{b}_j, \quad \text{s.t.} \quad B^TB = I, \tag{5.9}$$

i.e. $\{\mathbf{b}_j\}$ are the eigenvectors corresponding to the n largest eigenvalues of the covariance matrix:

$$C\mathbf{b}_j = \lambda_j \mathbf{b}_j, \tag{5.10}$$

where C is the covariance matrix:

$$C = \frac{1}{M}\sum_{i=1}^{M} \hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T = \frac{1}{M}\hat{X}\hat{X}^T, \tag{5.11}$$

and $\hat{X} = [\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N]$. Obviously $\mathbf{b}_j$ must lie in the subspace spanned by $[\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N]$, therefore we can write $\mathbf{b}_j = \hat{X}\boldsymbol{\alpha}_j$, and:

$$C\mathbf{b}_j = \lambda_j\mathbf{b}_j \Rightarrow \frac{1}{M}\hat{X}\hat{X}^T\hat{X}\boldsymbol{\alpha}_j = \lambda_j\hat{X}\boldsymbol{\alpha}_j, \text{ i.e. } \frac{1}{M}\hat{X}^T\hat{X}\boldsymbol{\alpha}_j = \lambda_j\boldsymbol{\alpha}_j. \tag{5.12}$$

Now we turn to the feature space Φ with feature mapping $\phi(\mathbf{x})$. Introduce the kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, and the centralized kernel:

$$\hat{k}(\mathbf{x}_i, \mathbf{x}_j) = \left(\phi(\mathbf{x}_i) - \frac{1}{M}\sum_{k=1}^{M}\phi(\mathbf{x}_k)\right)^T \left(\phi(\mathbf{x}_j) - \frac{1}{M}\sum_{k=1}^{M}\phi(\mathbf{x}_k)\right) \tag{5.13}$$

The PCA projection direction in the feature space can be obtained by calculating $\boldsymbol{\alpha}_j$ as the eigenvectors of the centralized kernel matrix:

$$\begin{aligned} &\hat{K} = \hat{X}^T\hat{X} = K - \mathbf{1}_M K - K\mathbf{1}_M + \mathbf{1}_M K \mathbf{1}_M, \\ &\tfrac{1}{M}\hat{K}\boldsymbol{\alpha}_j = \lambda_j \boldsymbol{\alpha}_j, \end{aligned} \tag{5.14}$$

where $\mathbf{1}_M$ is a $M \times M$ matrix, with all values equal to $1/M$. Furthermore, the requirement that ${\mathbf{b}_j}^T\mathbf{b}_j = 1$ transforms to:

$${\mathbf{b}_j}^T\mathbf{b}_j = {\boldsymbol{\alpha}_j}^T\hat{X}^T\hat{X}\boldsymbol{\alpha}_j = M\lambda_j {\boldsymbol{\alpha}_j}^T\boldsymbol{\alpha}_j = 1, \tag{5.15}$$

Therefore we normalize the vector $\boldsymbol{\alpha}_j$ such that its norm is $1/M\lambda_j$, and $M\lambda_j$ is the corresponding eigenvalue of $\hat{K}$. Then projecting a test instance to the principle direction $\mathbf{b}_j$ can be achieved by:

$$\phi(\mathbf{x})^T\mathbf{b}_j = \phi(\mathbf{x})^T\hat{X}\boldsymbol{\alpha}_j = \sum_{p=1}^{M}\left(\alpha_{j,p} - \frac{1}{M}\sum_{q=1}^{M}\alpha_{j,q}\right)k(\mathbf{x}_p, \mathbf{x}). \tag{5.16}$$

Compared with (5.8), the kernel PCA seeks a low dimensional subspace of the feature space Φ, such that the de-meaned training instances can be closely approximated by projections onto this subspace, i.e. solve the following problem:

$$\min_{B,V} \sum_{i=1}^{M}\left\|\hat{\phi}(\mathbf{x}_i) - B\mathbf{v}_i\right\|^2, \quad \text{s.t.} \quad B^TB = 1. \ , \tag{5.17}$$

where $\hat{\phi}(\mathbf{x}_i)$ is the de-meaned training instance in the feature space Φ, B is a matrix whose columns are $\{\phi(\mathbf{b}_1), ..., \phi(\mathbf{b}_1)\}$, which form an orthonormal basis of a subspace of the feature space Φ, and $\mathbf{v}_i$ is the combination coefficients for reconstructing $\hat{\phi}(\mathbf{x}_i)$. However, the image of the basis vectors in the original feature space, i.e. $\{\mathbf{b}_j\}$, cannot be obtained explicitly. Instead, $\phi(\mathbf{b}_j)$ is defined through the vector $\boldsymbol{\alpha}_j$.

5.2 Accelerate the kernel machines

We consider the computational cost of evaluating the linear classifier (5.2) in the feature space Φ. If the feature mapping $\phi(\mathbf{x})$ is not provided, and only the kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ is provided, the cost is usually $O(MN)$, assuming evaluating a kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ costs $O(N)$, where N is the dimension of the original feature space, and M is the number of basis vectors in the scoring function. The cost will be high if a large number of basis vectors are used. If the feature mapping $\phi(\mathbf{x})$ is provided at little cost, then the cost of the linear classifier is $O(D_\Phi)$, where D_Φ is the dimensionality of the feature space Φ. In this section we study how the efficiency of the kernel machine can be improved. Three classes of methods are discussed. The first class of methods directly considers approximating $f(\mathbf{x})$ in the original feature space. We discuss the work in [63], which applies to additive kernels that can be split into the sum over dimensions in the original feature space, for example, the histogram intersection kernel and the χ^2 kernel. The second class of methods consider finding an explicit feature map $\hat{\phi}(\mathbf{x})$, such that $\hat{\phi}(\mathbf{x}_i)^T \hat{\phi}(\mathbf{x}_j) \approx \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. The method in [64] is based on the concept of kernel signature, and applies to a range of kernels called the heterogeneous kernels. The method in [54] is based on a technique that is closely related to the kernel PCA in section 5.1.2, but chooses a sparse set of basis vectors to approximate the feature space Φ. The third class of methods [61, 60, 62] consider to represent $f(\mathbf{x})$ using a small number of basis vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_K\}$ in the original feature space, which may or may not be limited to the training instances. The basis set $\{\mathbf{b}_1, \ldots, \mathbf{b}_K\}$ is obtained along with solving the learning problem. For kernel SVM, the basis set is usually much smaller than the set of support vectors, i.e. the on-margin and in-margin training instances.

5.2.1 Approximate the scoring function

In this section we introduce the method proposed in [63], which applies to the additive kernels. An additive kernel in N-d feature space can be represented by the sum of kernels over its dimensions:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{p=1}^{N} k(x_{i,p}, x_{j,p}). \tag{5.18}$$

Then the scoring function can also be represented by the sum of functions operating on 1-D data:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m k(\mathbf{x}, \mathbf{x}_m) = \sum_{p=1}^{N} f(x_p), \text{ and } f(x_p) = \sum_{m=1}^{M} \alpha_m k(x_p, x_{m,p}). \quad (5.19)$$

Two most useful kernels in this category are the intersection kernel and the Chi-Square kernel:

$$\begin{aligned} k_{Inter}(x_i, x_j) &= \min(x_i, x_j), \\ k_{\chi^2}(x_i, x_j) &= x_i x_j / (x_i + x_j) \end{aligned} \quad (5.20)$$

In the following we focus on the intersection kernel. The scoring function of the intersection kernel for 1-D data is:

$$f(x) = \sum_{m=1}^{M} \alpha_m \min(x, x_m). \quad (5.21)$$

Directly evaluating (5.21) costs $O(M)$, since we have to compare the test instance x with all the basis $\{x_m\}$. However, (5.21) can be represented as:

$$f(x) = \sum_{x_m \le x} \alpha_m x_m + \sum_{x_m > x} \alpha_m x, \quad (5.22)$$

which is a piecewise linear function of x. Assume $\{x_m\}$ have been sorted in ascending order of values, and the partial sums have been calculated as:

$$S_{xm} = \sum_{j=1}^{m} \alpha_j x_j \text{ and } S_{\alpha m} = \sum_{j=1}^{m} \alpha_j, \quad (5.23)$$

then $f(x)$ can be evaluated using the following procedure:

- Find $j = \arg\max_j x_j \le x$, which can be achieved in $O(\log M)$ by binary search, since $\{x_j\}$ are sorted.
- Calculate $f(x) = S_{xj} + (S_{\alpha M} - S_{\alpha j})\, x$.

Therefore the 1-D intersection kernel scoring function can be evaluated exactly in $O(\log M)$, and for N-D intersection kernel the complexity is $O(N \log M)$. Furthermore, we can divide the value range of x into a number of bins with equal width, and approximate $f(x)$ in the k-th bin by a linear function $\hat{f}_k(x) = a_k x + b_k$. The parameter

a_k and b_k can be decided by least squares:

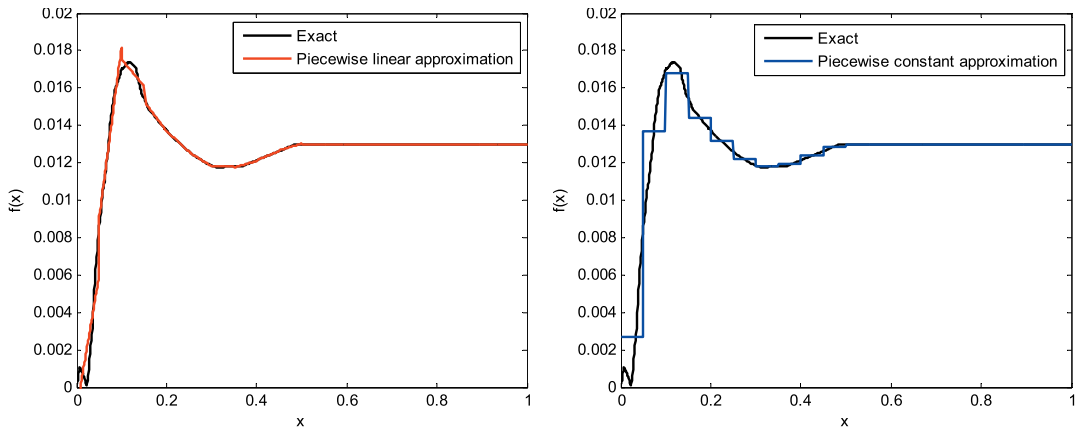
$$\min_{a_k,b_k} \int_{lb_k}^{ub_k} \left(f(x) - \hat{f}_k(x) \right)^2 dx, \tag{5.24}$$

where ub_k and lb_k are the upper bound and lower bound of the k-th bin. Then the approximated scoring function is:

$$\hat{f}(x) = \sum_k I(x \in bin_k) \hat{f}_k(x), \tag{5.25}$$

where $I(x \in bin_k) = 1$ if x is in the k-th bin, 0 otherwise. By using bins with equal width, $I(x \in bin_k)$ can be evaluated in $O(1)$, therefore the N-D kernel can be evaluated approximately in $O(N)$. The piecewise approximation can be applied to any additive kernel, and the approximation accuracy is high given sufficient number of pieces. Functions other than linear functions can also be employed in the approximation, e.g. polynomials or splines for better accuracy, or piecewise constant approximation for better speed.

In the end, we note that the piecewise approximation is limited to the additive kernels due to the curse of dimensionality, i.e. the number of pieces needed for good approximation accuracy increases exponentially fast with the growth of the dimensionality.



(a) (b)

Figure 5.1: (a) Piecewise linear approximation, and (b) Piecewise constant approximation of the scoring function using 20 pieces.

We show an example of the piecewise approximation in Fig. 5.1. The classifiers are trained using L-1 normalized HOG features, and thus the value range of each feature

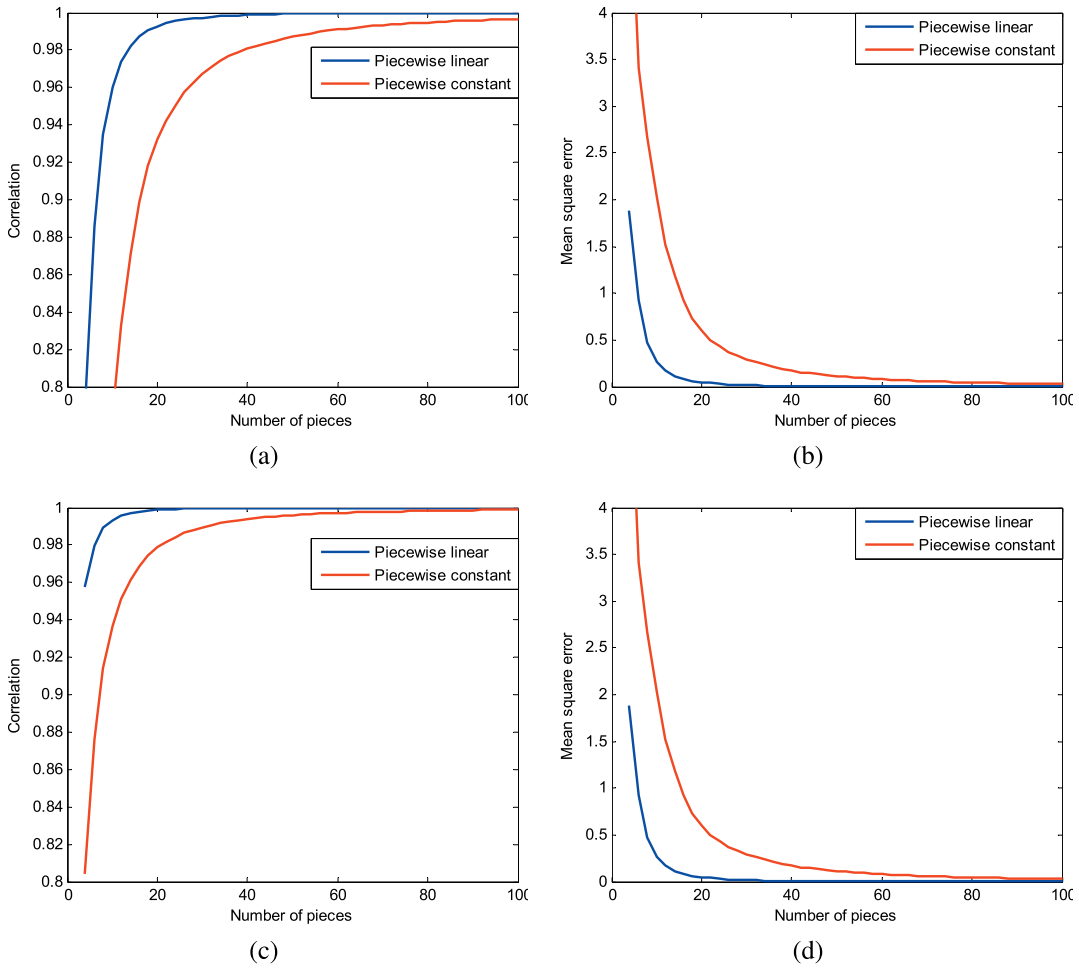


Figure 5.2: Approximation quality measured by (a)(c) the correlation between the exact functions and the approximated functions, and (b)(d) the mean square error between the exact functions and the approximated functions, for (a)(b) the intersection kernel and (c)(d) the χ^2 kernel.

is in $[0, 1]$. The quality of the approximation is measured by the correlation and mean square error between the exact function and the approximated function evaluated in $[0, 1]$ in Fig. 5.2 for the intersection kernel and the χ^2 kernel. It can be seen that the approximation accuracy can be very high using a small number of pieces. We evaluate the linear SVM, the intersection kernel SVM, and the χ^2 kernel SVM on the INRIA dataset. All methods are trained on the entire negative training set, including millions of windows from the 1,218 negative training images. The n-slack cutting plane algorithm is used for quickly identifying the support vectors. The Detector Error Tradeoff (DET) curves are shown in Fig. 5.3. The horizontal axis shows the False Positive Per Window rate as measured using all windows (tens of millions) in the

negative test images not containing any human. We can see both the intersection kernel and the χ^2 kernel improves over the linear kernel. The Recall rate for $FPPW = 10^{-4}$ is improved by about 3%, and the FPPW is reduced by half for 0.90 recall.

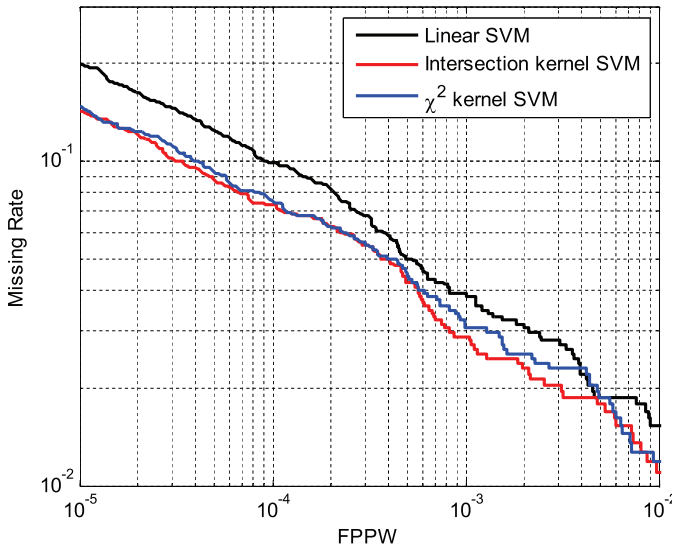


Figure 5.3: Detection Error Tradeoff curves for the linear kernel, the intersection kernel, and the χ^2 kernel using piecewise linear approximation.

5.2.2 Explicit feature map

Although the scoring function can be approximated with high accuracy by the piecewise linear functions, the classifier need to be trained as a kernel SVM, usually by solving the dual problem, and the training complexity for non-linear kernel SVM usually scales as $O(M^2)$ with the number of training instances M. On the other hand, efficient $O(M)$ methods exist for linear SVM. If the explicit feature map $\phi(x)$ can be found for the kernel $k(x_i, x_j)$, we can directly train the classifier as a linear SVM. Two techniques for building an approximated explicit feature map are discussed in this section. The first technique is constrained to kernels satisfying some particular properties, and the second technique applies to arbitrary kernels, but is usually less efficient than the first one.

Kernel signature technique for homogeneous kernels

In the following we limit our discussion to kernels with 1-D features. The technique can be generalized to N-D features for additive kernels, i.e. $k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} k(x_j, y_j)$, by applying the technique to each dimension. We introduce the method in [64], which

find an approximated explicit feature map $\phi(x)$, such that evaluating the scoring function (5.2) costs $O(D_\Phi)$, where D_Φ is the dimension of the feature space Φ. The method applies to kernels that can be represented by a *kernel signature function*, and satisfying some additional requirements. In the following we take the class of homogeneous kernels for example. For 1-D feature, a kernel is γ homogeneous if it has the following property:

$$\forall c \geq 0,\, k_h(cx, cy) = c^\gamma k_h(x, y). \tag{5.26}$$

Choosing $c = 1/\sqrt{xy}$, the kernel can be represented as:

$$k_h(x, y) = c^{-\gamma} k_h(cx, cy) = (xy)^{\frac{\gamma}{2}} k_h\left(\sqrt{y/x}, \sqrt{x/y}\right) = (xy)^{\frac{\gamma}{2}} \mathcal{K}(\log y - \log x). \tag{5.27}$$

$\mathcal{K}(\lambda) = k_h\left(e^{\lambda/2}, e^{-\lambda/2}\right)$ is called the kernel signature function. The kernel signature is a positive definite function. By the Bochener's theorem, it is the Fourier transform of a function $\kappa(\omega)$, called its spectrum:

$$\mathcal{K}(\lambda) = \int_{-\infty}^{+\infty} e^{-i\omega\lambda} \kappa(\omega)\, d\omega, \text{ and } \kappa(\omega) = \frac{1}{2\pi} \int e^{i\omega\lambda} \mathcal{K}(\lambda)\, d\lambda. \tag{5.28}$$

Therefore:

$$\begin{aligned} k(x, y) &= (xy)^{\frac{\gamma}{2}} \mathcal{K}(\lambda) = (xy)^{\frac{\gamma}{2}} \int_{-\infty}^{+\infty} e^{-i\omega\lambda} \kappa(\omega)\, d\omega,\, \lambda = \log \frac{y}{x} \\ &= \int_{-\infty}^{+\infty} \left(e^{-i\omega \log x} \sqrt{x^\gamma \kappa(\omega)}\right)^* \left(e^{-i\omega \log y} \sqrt{y^\gamma \kappa(\omega)}\right) d\omega, \end{aligned} \tag{5.29}$$

where the superscript * indicates the complex conjugate. Based on (5.29), an infinite dimensional feature map indexed by the continuous variable ω can be derived for the homogeneous kernel:

$$\phi_\omega(x) = e^{-i\omega \log x} \sqrt{x^\gamma \kappa(\omega)} \tag{5.30}$$

The kernel signature function and its spectrum for the intersection kernel and the χ^2 kernel are listed in table 5.1. The kernel signatures are displayed in Fig. 5.4a, and their spectrums in 5.4b. The continuous feature map functions for $x = 0.9$ and $x = 1$ are displayed in Fig. 5.4c and 5.4d, respectively.

To derive a finite dimensional feature map, we try to sample and truncate the spectrum $\kappa(\omega)$, while ensuring $\mathcal{K}(\lambda)$ can be approximated with high accuracy in a neighborhood near the origin. Sampling the spectrum, i.e. multiplying $\kappa(\omega)$ with an impulse train, is equivalent to convolving $\mathcal{K}(\lambda)$ with an impulse train, i.e. repeating

kernel	$k(x,y)$	signature $\mathcal{K}(\lambda)$	$\kappa(\omega)$
intersection	$\min(x,y)$	$\exp(-\lvert\lambda\rvert/2)$	$\frac{2}{\pi(1+4\omega^2)}$
χ^2	$(2xy)/(x+y)$	$\operatorname{sech}(\lambda/2)$	$\operatorname{sech}(\pi\omega)$

Table 5.1: Kernel, kernel signature, and the spectrum for the intersection kernel and the χ^2 kernel.

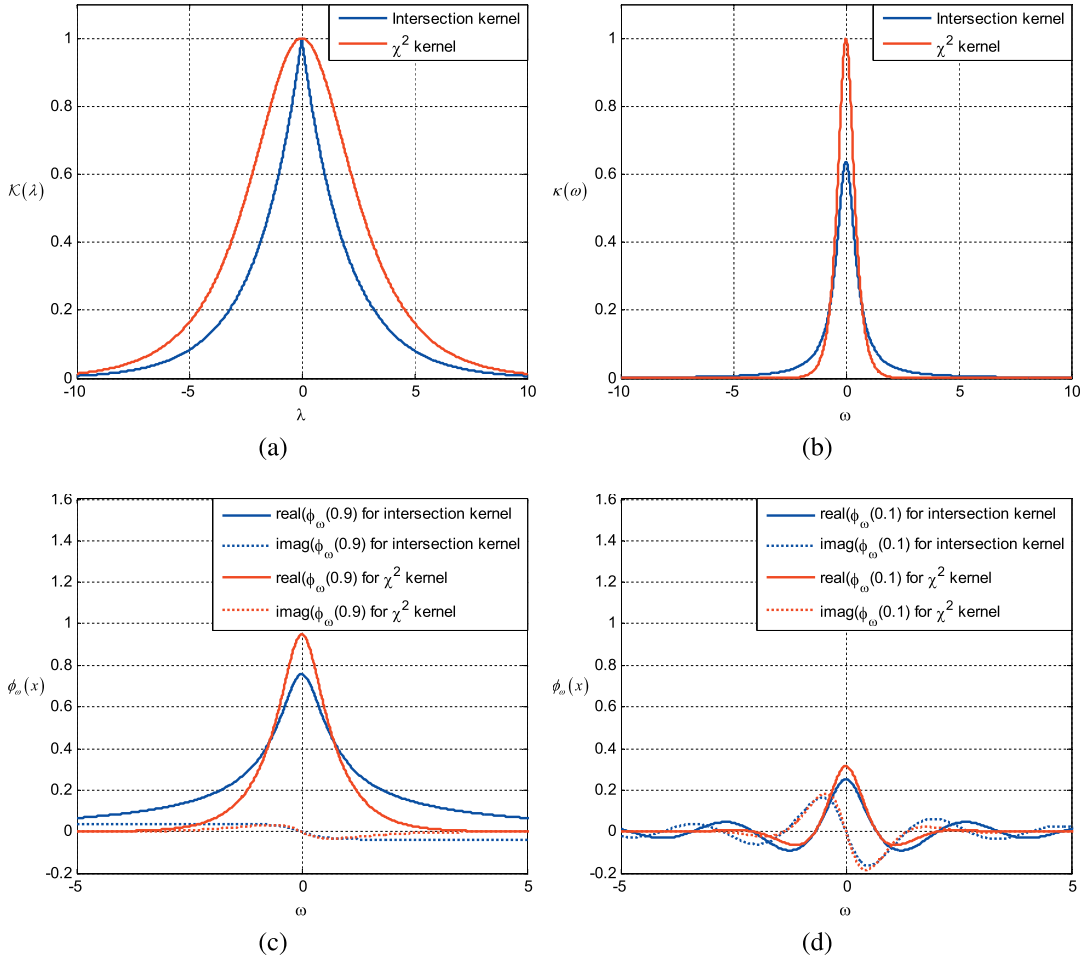


Figure 5.4: (a) Kernel signature functions for the intersection kernel and the χ^2 kernel. (b) Spectrum of the kernel signature functions for the intersection kernel and the χ^2 kernel. (c) Continuous feature map function evaluated at $x = 0.9$. (d) Continuous feature map function evaluated at $x = 0.1$.

$\mathcal{K}(\lambda)$:

$$\begin{aligned} \widehat{\mathcal{K}}(\lambda) &= \underset{\Lambda}{per}\,\mathcal{K}(\lambda) = \mathcal{K}(\lambda) * \Delta_\Lambda(\lambda) \\ \hat{\kappa}_j &= L\kappa(jL) \end{aligned} \tag{5.31}$$

$*$ means convolution, $\Delta_\Lambda(\lambda)$ is the impulse train in time domain with period Λ, and

$L = 2\pi/\Lambda$ is the corresponding sampling period in the frequency domain for κ. Convolving with $\Delta_\Lambda(\lambda)$ will cause distortion for $\mathcal{K}(\lambda)$. In order to have good approximation for $\mathcal{K}$ near the origin, we require $\mathcal{K}(\lambda)$ to decay fast in the time domain, and the period Λ of the time domain impulse train be large enough, i.e. the sampling in frequency domain be sufficiently dense, such that the approximation error due to aliasing is low. If $\mathcal{K}(\lambda)$ does not decay sufficiently fast, to maintain good approximation accuracy of $\mathcal{K}(\lambda)$ near the origin, a rectangular window of width Λ is applied to $\mathcal{K}(\lambda)$:

$$W(\lambda) = rect(\lambda/\Lambda) \tag{5.32}$$

$$\hat{\mathcal{K}}(\lambda) = \underset{\Lambda}{per}\, W(\lambda)\mathcal{K}(\lambda) = (W(\lambda)\mathcal{K}(\lambda)) * \Delta_\Lambda(\lambda) \tag{5.33}$$

Then $\hat{\kappa}_j = L(\kappa * w)(jL)$, where w is the inverse Fourier transform of $W(\lambda)$. The rectangular window $W(\lambda) = rect(\lambda/\Lambda)$ is not a positive definite function, and $w(\omega)$ contains negative values. Therefore we truncate $w(\omega)$ to be non-negative to ensure positive definiteness of $\hat{\mathcal{K}}(\lambda)$.

The approximated infinite dimensional discrete feature map $\{\phi_j(x), j = 0, ..., +\infty\}$ can be obtained from the continuous feature map (5.30) as follows:

$$\phi_j(x) = \begin{cases} \sqrt{x^\gamma \hat{\kappa}_0} & j = 0, \\ \cos\left(\frac{j+1}{2} L \log x\right)\sqrt{2x^\gamma \hat{\kappa}_{(j+1)/2}} & j > 0 \text{ odd}, \\ \sin\left(\frac{j}{2} L \log x\right)\sqrt{2x^\gamma \hat{\kappa}_{j/2}} & j > 0 \text{ even}. \end{cases} \tag{5.34}$$

It can be verified that :

$$\begin{aligned} \langle \phi(x), \phi(y) \rangle &= (xy)^{\gamma/2} \sum_{j=-\infty}^{+\infty} \hat{\kappa}_j e^{-ijL\log\frac{y}{x}} \\ &= (xy)^{\gamma/2}\hat{\mathcal{K}}\left(\log\frac{y}{x}\right) \approx (xy)^{\gamma/2}\mathcal{K}\left(\log\frac{y}{x}\right) = k_h(x,y), \end{aligned} \tag{5.35}$$

i.e. $\phi(x)$ is an approximated feature map of the homogeneous kernel $k_h(x,y)$. Furthermore, the sequence $\hat{\kappa}_j$ is truncated, keeping only those terms $|j| \leq (n-1)/2$, where n is an odd number specifying how many terms in $\{\phi_j(x)\}$ are kept in the finite dimensional feature map. For the truncation to work well, the spectrum $\kappa(\omega)$ should also have most of its energy in the low frequency range.

To sum up, the following issues are important in applying this technique to find an explicit approximated feature map for a kernel:

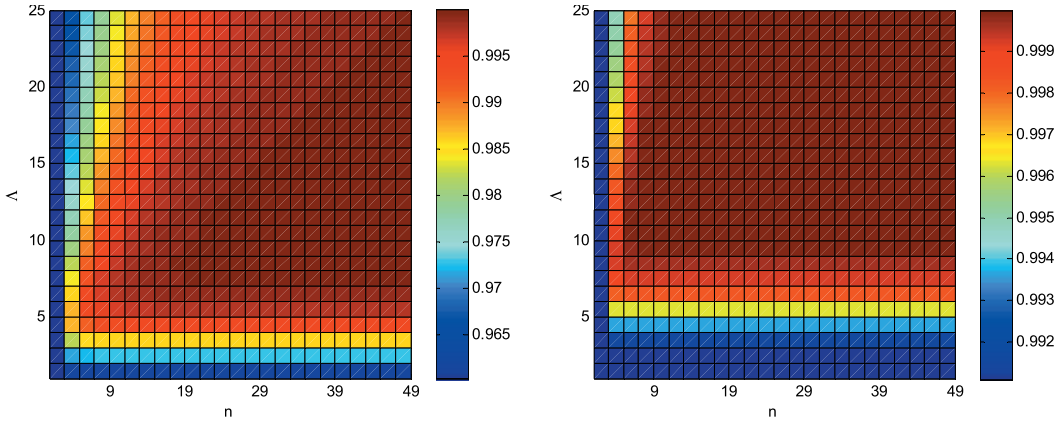
1. A signature function must exist for the kernel.

2. The signature function is a fast decaying function such that we can sample its spectrum, i.e. convolving the signature function with a impulse train, while still maintaining a good approximation of the signature function near the origin. Otherwise, we need to apply a window to the signature function to avoid aliasing.

3. The spectrum of the signature function concentrates most of its energy in the low frequency part, such that truncating the spectrum will not cause much distortion.

4. The period Λ of the time-domain impulse train and the order n of the approximation need to be carefully tuned for good accuracy.

1 is satisfied by all homogeneous kernels. As shown in Fig. 5.4, 2 and 3 are satisfied by both the χ^2 kernel and the intersection kernel, both of which are homogeneous kernels. We measure the approximation quality by the correlation between the exact kernel and the approximated kernel using the explicit feature map. The correlation for various Λ and n are displayed in Fig. 5.5. Very high accuracy can be achieved for the χ^2 kernel using $n = 3$, while the intersection kernel needs larger n to achieve good accuracy, since it has a non-differentiable point in the kernel signature function and thus its spectrum spans a wider range. To have a direct impression of the approximation accuracy, we show the ratio between the approximated kernel and the exact kernel evaluated for $x \in [0, 1]$ and $y \in [0, 1]$ as a color images in Fig. 5.6 for various n. We can see the error concentrates near the boundary of the domain.



(a) (b)

Figure 5.5: Correlations between the exact kernel and the kernel using the explicit feature map for various n and Λ. (a) Intersection kernel; (b) χ^2 kernel.

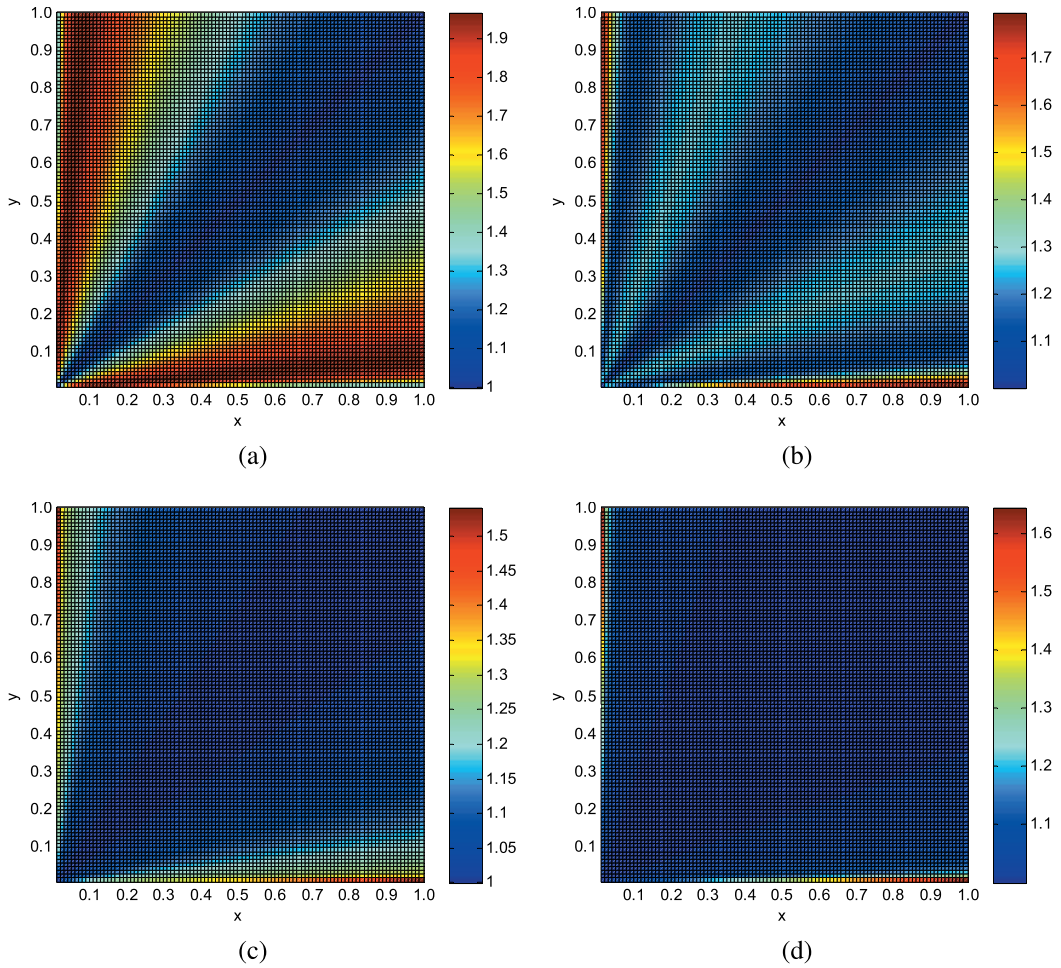


Figure 5.6: Ratio between the approximated kernel and the exact kernel. (a) Intersection kernel using $\Lambda = 10$ and $n = 3$; (b) Intersection kernel using $\Lambda = 10$ and $n = 7$; (c) χ^2 kernel using $\Lambda = 10$ and $n = 3$; (d) χ^2 kernel using $\Lambda = 10$ and $n = 7$.

Sparse basis set feature subspace

In this part we introduce the method in [54], which is called the kernel features by the authors, but essentially approximates the feature space Φ using a subspace whose basis corresponds to a sparse set of feature vectors in the original feature space, therefore we call the method "sparse basis set feature subspace".

We want to find a low dimensional subspace of the feature space Φ, such that a training instance $\phi(\mathbf{x})$ can be approximated with high accuracy by the projection in this subspace. Let the columns of matrix B be the basis vectors of this subspace, $B = [\phi(\mathbf{b}_1), ..., \phi(\mathbf{b}_K)]$, and $\{\mathbf{b}_k\}$ are their image in the original feature space. Then

a low-dimensional approximation $\mathbf{v}$ of instance $\phi(\mathbf{x})$ can be calculated by:

$$\underset{\mathbf{v}}{\operatorname{argmin}} \quad \|\phi(\mathbf{x}) - B\mathbf{v}\|^2 \tag{5.36}$$

The solution can be obtained in closed form as $\mathbf{v} = \left(B^T B\right)^{-1}\left(B^T \phi(\mathbf{x})\right)$. Denote $G = B^T B$, then G is a $K \times K$ matrix, $G_{ij} = \phi(\mathbf{b}_i)^T \phi(\mathbf{b}_j) = k(\mathbf{b}_i, \mathbf{b}_j)$. Denote $\mathbf{K} = B^T \phi(\mathbf{x})$, then $\mathbf{K}$ is a $K \times 1$ vector, $K_k = \phi(\mathbf{b}_k)^T \phi(\mathbf{x}) = k(\mathbf{b}_k, \mathbf{x})$. Then $\phi(\mathbf{x})$ can be approximated by:

$$\hat{\phi}(\mathbf{x}) = B\mathbf{v} = BG^{-1}\mathbf{K}. \tag{5.37}$$

Since $\hat{\phi}(\mathbf{x}_i)^T \hat{\phi}(\mathbf{x}_j) = \mathbf{K}_i^T G^{-1} \mathbf{K}_j$, where $\mathbf{K}_i = B^T \phi(\mathbf{x}_i)$ and $\mathbf{K}_j = B^T \phi(\mathbf{x}_j)$, therefore the approximated explicit feature map can also be represented as:

$$\hat{\phi}(\mathbf{x}) = G_{-1/2}\mathbf{K}, \tag{5.38}$$

where $G_{-1/2}$ is the solution of $G_{-1/2}^T G_{-1/2} = G^{-1}$, and can be obtained by Cholesky decomposition of G^{-1}.

However, what remains is how to construct the basis set B. We look for the basis vectors $\{\mathbf{b}_i\}$to minimize the approximation error for the training data:

$$\min_{V, \{\mathbf{b}_i\}} R(V, B) = \sum_{i=1}^{M} \|\phi(\mathbf{x}_i) - B\mathbf{v}_i\|^2. \tag{5.39}$$

Substitute $\mathbf{v}_i = \left(B^T B\right)^{-1}\left(B^T \phi(\mathbf{x}_i)\right)$ into the problem, the problem transforms to:

$$\max_{B} \quad R(B) = \sum_{i=1}^{M} \hat{k}(\mathbf{x}_i, \mathbf{x}_i), \tag{5.40}$$

where $\hat{k}(\mathbf{x}_i, \mathbf{x}_i) = \hat{\phi}(\mathbf{x}_i)^T \hat{\phi}(\mathbf{x}_i) = \mathbf{K}_i^T G^{-1} \mathbf{K}_i$. The problem can be solved by gradient-based algorithms, e.g. the LBFGS algorithm, which is a Quasi-Newton method that uses a low-rank approximation of the Hessian based on the gradients, and does not need to explicitly store the Hessian, making it suitable for solving high dimensional problems ($N \times K$ dimensions in our problem). The gradient of $R(B)$ can be calculated as follows:

$$\frac{\partial R}{\partial \mathbf{b}_k} = \sum_{i=1}^{M} \left[2\left(\frac{\partial \mathbf{K}_i}{\partial \mathbf{b}_k}\right)^T G^{-1}\mathbf{K}_i + \mathbf{K}_i^T G^{-1} \frac{\partial G}{\partial \mathbf{b}_k} G^{-1}\mathbf{K}_i \right] \tag{5.41}$$

Now we consider the computational complexity for calculating the gradient, for M

training instances, K basis vectors, and N dimensional original feature space. Computing G costs $O(NK^2)$, and computing $\mathbf{K}_i$ for all $i \in \{1, ..., M\}$ costs $O(NMK)$. Then $G^{-1}\mathbf{K}_i$ can be computed for all i in $O(MK^2)$. Let n be the index of a feature, $\partial\mathbf{K}_i/\partial b_{k,n}$ is a vector of length K, in which only the k-th element is non-zero. Therefore each $(\partial\mathbf{K}_i/\partial b_{k,n})^T G^{-1}\mathbf{K}_i$ can be computed in $O(1)$, and $(\partial\mathbf{K}_i/\partial\mathbf{b}_k)^T G^{-1}\mathbf{K}_i$ can be computed in $O(N)$, and the first term in the derivative can be computed in $O(NMK)$ for all $\mathbf{b}_k$. $\partial G/\partial b_{k,n}$ is a matrix with K rows and K columns, but only the k-th row and the k-th column are non-zero. Therefore the second term in the derivative (5.41) can be computed in $O(NMK)$ for each $\mathbf{b}_k$, and $O(NMK^2)$ for all k. Overall, the computational cost of evaluating the gradient is $O(NMK^2)$.

The problem formulation (5.39) is very similar to the kernel PCA problem (5.17), both minimizing the square error of the approximation. But here the columns of B are limited to examples in the original feature space, i.e. $\{\phi(\mathbf{b}_i)\}$, and are not required to be orthonormal, while in (5.17) the columns of B can be arbitrary linear combination of $\{\phi(\mathbf{x}_i)\}$, and the combination coefficient $\boldsymbol{\alpha}$ is usually not sparse. Therefore this approach reduces the cost of the linear classifier in the feature space from $O(NM)$ to $O(NK)$, where N is the dimension of the original feature space, M is the size of the original basis vector set, e.g. the full training set for kernel PCA, the support vectors for kernel SVM, and K is the number of basis vectors for approximating the feature space Φ. Usually M grows when more training data are given, but K can be decided as a constant value, and hence the classifier scales better when a large training data is available.

For additive kernels $k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} k(x_j, y_j)$, the technique can be applied separately to each dimension. Compared with the explicit feature map based on the kernel signatures, this sparse basis set method can be used for arbitrary kernels, but the downside is that the approximation accuracy depends on the training data.

5.2.3 Sparse kernel machines

The cost of evaluating a kernel scoring function can also be reduced by using a limited number of basis vectors $\{\mathbf{b}_i\}_{i=1}^{K}$ to represent the scoring function f:

$$f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i \phi(\mathbf{b}_i)^T \phi(\mathbf{x}) = \sum_{i=1}^{K} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{5.42}$$

The basis set $\{\mathbf{b}_i\}_{i=1}^K$ is much smaller than the training set $\{\mathbf{x}_i\}_{i=1}^M$, and the complexity of the classifier can be explicitly controlled by the basis number K. Instead of allowing $\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}_i)$ to be any vector in the span of the training instances, $\mathbf{w}$ is constrained to be in the subspace spanned by this basis set. The sparse basis set subspace approach discussed in the last section essentially achieves this goal, but constructing the basis set and learning the classifier are two separate stages. Therefore the basis set is selected for good representation of the data rather than for good classification performance. In this section, we decide the basis set during the learning process, such that the basis set is directly optimized for the learning task. Three methods for building the sparse kernel machine are discussed for kernel SVM. The first method from [61] uses a fixed number of basis vectors, and optimizes the basis vectors to improve the SVM objective. The second approach [60] sequentially introduces the basis vectors in a greedy manner, selecting the basis vector that mostly improves the SVM objective in each round. The last approach [62] adapts the cutting plane training in linear SVM for training kernel SVM, by approximating the cutting planes using a small number of basis vectors.

Reduced kernel SVM with basis vector refinement

Approximating the model parameter $\mathbf{w}$ using a reduced set of basis vectors $\{\mathbf{b}_i\}$ has been proposed in [113], which learns the SVM and then finds the basis set to minimize $\|\mathbf{w} - \hat{\mathbf{w}}\|_2^2$, where $\hat{\mathbf{w}}$ is the projection of $\mathbf{w}$ in the subspace spanned by the basis vectors. A direct approach for finding the basis set along with learning the SVM is proposed in [61], which we briefly introduce in this part. For now we consider the scoring function without the intercept term b, i.e. $f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$. By explicitly formulating $\mathbf{w}$ as in the span of the training instances $\{\phi(\mathbf{x}_i)\}$, we formulate the SVM learning problem as follows:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^M \xi_i \\ \text{s.t.} \quad & \mathbf{w} = \sum_{i=1}^M \alpha_i \phi(\mathbf{x}_i), \\ & y_i \mathbf{w}^T \phi(\mathbf{x}_i) \geq 1 - \xi_i, i = 1, ..., M. \\ & \xi_i \geq 0, i = 1, ..., M. \end{aligned} \tag{5.43}$$

The solution is identical to the original kernel SVM problem (5.3), and the solution $\boldsymbol{\alpha}$ is a sparse vector such that only those support vectors (in-margin and on-margin training instances) contribute to the separating hyperplane $\mathbf{w}$.

Instead of representing $\mathbf{w}$ using the full training set $\{\mathbf{x}_i\}$, a reduced set of basis

vectors $\{\mathbf{b}_i\}$ can be used, such that $\mathbf{w} = \sum_{i=1}^{K} \alpha_i \phi(\mathbf{b}_i)$. The basis set $\{\mathbf{b}_i\}$ could be a subset of the training set, or a different set of instances. Then the learning problem becomes:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \\ \text{s.t.} \quad & \mathbf{w} = \sum_{i=1}^{K}\alpha_i\phi(\mathbf{b}_i), \\ & y_i\mathbf{w}^T\phi(\mathbf{x}_i) \geq 1-\xi_i, i = 1,...,M, \\ & \xi_i \geq 0, i = 1,...,M. \end{aligned} \tag{5.44}$$

In contrast to the original kernel SVM, for the SVM using a reduced set of basis vectors, the weight $\boldsymbol{\alpha}$ of the basis vectors usually is not sparse. The dual problem is:

$$\begin{aligned} \max_{\boldsymbol{\beta}} \quad & \tfrac{1}{2}\boldsymbol{\beta}^T K_{BX}^T G^{-1} K_{BX}\boldsymbol{\beta} - \sum_{i=1}^{M}\beta_i y_i \\ \text{s.t.} \quad & 0 \leq \beta_i y_i \leq C. \end{aligned} \tag{5.45}$$

If we consider the scoring function with a intercept term b, i.e. $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b$, the only difference to the dual problem is to add a constraint $\sum_{i=1}^{M}\beta_i = 0$. K_{BX} is a matrix with K rows and M columns, each column corresponds to a training instance $\mathbf{x}_j$ and each row corresponds to a basis vector $\mathbf{b}_i$, i.e. $K_{BX,ij} = k(\mathbf{b}_i, \mathbf{x}_j)$. In the following we denote $\mathbf{K}_i$ as the i-th column of K_{BX}. G is a $K \times K$ matrix, and $G_{ij} = k(\mathbf{b}_i, \mathbf{b}_j)$. Compare this formulation with the standard SVM dual problem, it is easy to see that this problem is equivalent to a SVM using kernel matrix $\hat{K} = K_{BX}^T G^{-1} K_{BX}$. $\hat{K}$ is a $M \times M$ matrix, and $\hat{K}_{ij} = \mathbf{K}_i^T G^{-1}\mathbf{K}_j$. This result is essentially identical to the case we have studied in the sparse basis set feature subspace method, where an explicit feature map can be constructed as:

$$\hat{\phi}(\mathbf{x}_i) = G_{-1/2}\mathbf{K}_i, \text{ where } G^{-1} = {G_{-1/2}}^T G_{-1/2}. \tag{5.46}$$

Therefore we can construct the explicit feature map and then train a linear SVM using $\hat{\phi}(\mathbf{x}_i)$. The classifier is

$$f(\mathbf{x}) = \hat{\mathbf{w}}^T\hat{\phi}(\mathbf{x}) = \hat{\mathbf{w}}^T G_{-1/2}\mathbf{K}_i, \text{ where } \hat{\mathbf{w}} = G_{-1/2}K_{BX}\boldsymbol{\beta}. \tag{5.47}$$

Then the weight of the basis vectors can be recovered as follows:

$$f(\mathbf{x}_i) = \mathbf{w}^T\phi(\mathbf{x}_i) = \boldsymbol{\alpha}^T\mathbf{K}_i = \hat{\mathbf{w}}^T G_{-1/2}\mathbf{K}_i \Rightarrow \boldsymbol{\alpha} = G_{-1/2}^T\hat{\mathbf{w}} = G^{-1}K_{BX}\boldsymbol{\beta} \tag{5.48}$$

Instead of constructing the basis set $\{\mathbf{b}_i\}$ by solving (5.40) as in the sparse basis set feature space method, the basis set is learned to optimize the SVM objective, by solving the following problem:

$$\min_{B} \quad R(B), \tag{5.49}$$

where $R(B)$ is the optimal value of problem (5.44). Gradient-based methods can be employed to solve (5.49), e.g. the LBFGS algorithm. For a particular $\boldsymbol{\beta}$, the gradient of $R(B)$ can be computed according to the dual problem (5.45), since the solution of the dual problem is identical to the primal problem:

$$\frac{\partial R}{\partial \mathbf{b}_k} = \frac{1}{2}\boldsymbol{\beta}^T \frac{\partial \hat{K}}{\partial \mathbf{b}_k}\boldsymbol{\beta}. \tag{5.50}$$

The gradient of $\hat{K}$ is:

$$\frac{\partial \hat{K}_{ij}}{\partial \mathbf{b}_k} = \frac{\partial \mathbf{K}_i}{\partial \mathbf{b}_k} G^{-1}\mathbf{K}_j + \mathbf{K}_i^T G^{-1}\frac{\partial \mathbf{K}_j}{\partial \mathbf{b}_k} + \mathbf{K}_i^T \frac{\partial G^{-1}}{\partial \mathbf{b}_k}\mathbf{K}_j \tag{5.51}$$

$$\frac{\partial G^{-1}}{\partial \mathbf{b}_k} = -G^{-1}\frac{\partial G}{\partial \mathbf{b}_k}G^{-1}. \tag{5.52}$$

Therefore the gradient of R can be computed as:

$$\frac{\partial R}{\partial \mathbf{b}_k} = \sum_{i,j}\beta_i \frac{\partial \hat{K}_{ij}}{\partial \mathbf{b}_k}\beta_j = 2\sum_{i=1}^{M}\beta_i \left(\frac{\partial \mathbf{K}_i}{\partial \mathbf{b}_k}\right)^T G^{-1}K_{BX}\boldsymbol{\beta} + \left(G^{-1}K_{BX}\boldsymbol{\beta}\right)^T \frac{\partial G}{\partial \mathbf{b}_k}G^{-1}K_{BX}\boldsymbol{\beta}, \tag{5.53}$$

which can be accomplished in $O(NMK)$. In implementation, we initialize the basis set B using KMeans clustering on the training data, and then refine the basis set using the LBFGS algorithm.

Forward basis selection

Instead of refining the basis set B for which the size $K = |B|$ is pre-defined, the basis vectors can sequentially introduced to optimize the SVM objective, as done in [60]. For the current basis set $\{\mathbf{b}_i\}_{i=1}^{K}$, the SVM learning problem is:

$$\min_{\boldsymbol{\alpha}} \quad \tfrac{1}{2}\boldsymbol{\alpha}^T G\boldsymbol{\alpha} + C\sum_{i=1}^{M}\max\left(0, 1 - y_i\sum_k \alpha_k k(\mathbf{b}_k, \mathbf{x}_i)\right), \tag{5.54}$$

where G is the kernel matrix for the selected basis vectors, $G_{ij} = k(\mathbf{b}_i, \mathbf{b}_j)$. To select a new basis vector $\mathbf{b}_{K+1}$ into the basis set, we fix the weight vector $\boldsymbol{\alpha}$ for the basis set $\{\mathbf{b}_i\}_{i=1}^K$ which are already in the problem, and solve for the weight α of the new basis vector $\mathbf{b}$ by solving the following one-variable problem:

$$\begin{aligned}\min_{\alpha} \quad & \tfrac{1}{2}\begin{bmatrix}\boldsymbol{\alpha}^T & \alpha\end{bmatrix}\begin{bmatrix}G & \mathbf{K}_{Bb} \\ \mathbf{K}_{Bb}^T & k(\mathbf{b},\mathbf{b})\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha} \\ \alpha\end{bmatrix} \\ & +C\sum_{i=1}^{M}\max\left(0, 1-y_i\left(\alpha k(\mathbf{b},\mathbf{x}_i)+\sum_{j=1}^{K}\alpha_j k(\mathbf{b}_j,\mathbf{x}_i)\right)\right)\end{aligned} \tag{5.55}$$

which can be reduced to the following problem:

$$\begin{aligned}\min_{\alpha} \quad & R(\alpha) = \tfrac{1}{2}\alpha k(\mathbf{b},\mathbf{b})\,\alpha + \boldsymbol{\alpha}^T\mathbf{K}_{Bb}\alpha \\ & +C\sum_{i=1}^{M}\max(0, 1-m_i-y_i\alpha k(\mathbf{b},\mathbf{x}_i))\end{aligned} \tag{5.56}$$

where $\mathbf{K}_{Bb}$ is a length k vector, $K_{Bb,i} = k(\mathbf{b}_i, \mathbf{b})$, and $m_i = \sum_{j=1}^{K}\alpha_j k(\mathbf{b}_j, \mathbf{x}_i)$ is the score of a training instance using the current model. The problem is piecewise quadratic in α_{K+1}. We can solve it using Newton's method. For the current value of α, the Hessian of $R(\alpha)$ is $k(\mathbf{b},\mathbf{b})$, while the gradient is:

$$\frac{\partial R(\alpha)}{\partial \alpha} = k(\mathbf{b},\mathbf{b})\,\alpha + \boldsymbol{\alpha}^T\mathbf{K}_{Bb} - C\sum_{i=1}^{M}\pi_i y_i k(\mathbf{b},\mathbf{x}_i), \tag{5.57}$$

where π_i is an indicator, $\pi_i = 0$ if $y_i\alpha k(\mathbf{b},\mathbf{x}_i) \geq 1 - m_i$, and $\pi_i = 1$ otherwise. Then a line search can be performed along the Newton direction to minimize the objective value. Among all possible choices of $\mathbf{b}$, we select the one that minimizes the loss $R(\alpha)$.

Cutting plane sparse kernel SVM

Based on the 1-slack cutting plane algorithm for solving linear SVMs [106], a method is proposed in [62] to adapt the 1-slack cutting plane algorithm for training kernel SVMs, by approximating the cutting planes using a small number of basis vectors.

Recall that the 1-slack cutting plane method for linear SVM solves the following

problem in iteration T:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi \\ \text{s.t.} \quad & \mathbf{w}^T \sum_{i=1}^{M} \pi_{ti} y_i \mathbf{x}_i \geq \sum_{i=1}^{M} \pi_{ti} - \xi, t = 1, ..., T, \\ & \xi \geq 0. \end{aligned} \tag{5.58}$$

The indicator vectors $\{\boldsymbol{\pi}_t\}$ are found according to the cutting planes of the unconstrained problem:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M} \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) \tag{5.59}$$

For a $\mathbf{w}_t$, the indicator vector is decided as follows:

$$\pi_{ti} = \begin{cases} 1 & y_i \mathbf{w}_t^T \mathbf{x}_i \leq 1 \\ 0 & y_i \mathbf{w}_t^T \mathbf{x}_i > 1 \end{cases}, \tag{5.60}$$

Denote $\Psi_t = \sum_{i=1}^{M} \pi_{ti} y_i \mathbf{x}_i$ and $\Delta_t = \sum_{i=1}^{M} \pi_{ti}$, the dual problem can be derived as:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\tfrac{1}{2}\boldsymbol{\alpha}^T H \boldsymbol{\alpha} + \boldsymbol{\alpha}^T \boldsymbol{\Delta} \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, ..., T, \\ & \mathbf{1}^T \boldsymbol{\alpha} \leq C, \end{aligned} \tag{5.61}$$

where H is a $T \times T$ matrix, $H_{ij} = \Psi_i^T \Psi_j$. The solution of (5.58) satisfies :

$$\mathbf{w} = \sum_{i=1}^{T} \alpha_i \Psi_i = \sum_{i=1}^{m} \bar{\alpha}_m \bar{\Psi}_m, \tag{5.62}$$

where $\left\{\bar{\Psi}_1, ..., \bar{\Psi}_m\right\}$ is a subset of $\{\Psi_t\}$ satisfying $\mathbf{w}^T \Psi_t = \Delta_t - \xi$, i.e. is the set of active constraints. The solution $\mathbf{w}$ is in the subspace spanned by $\left\{\bar{\Psi}_1, ..., \bar{\Psi}_m\right\}$:

$$\mathbf{w} \in \mathcal{W} = span\left(\bar{\Psi}_1, ..., \bar{\Psi}_m\right). \tag{5.63}$$

The cutting plane training is summarized in Algorithm 4.

To evaluate $\boldsymbol{\pi}_t$ and add a cutting plane, we need to evaluate $\mathbf{w}_t^T \mathbf{x}$ for all training instances. Then to solve the problem (5.61), we need to update the matrix H, add a new row and column corresponding to the new cutting plane Ψ_T, i.e. calculating $\Psi_T^T \Psi_t$

Algorithm 4 Train linear SVM by solving (5.59) using 1-slack cutting plane method

Initialization: $\mathbf{w}_1 = 1$.
for $T = 1$ to T_{max} **do**
  Add a cutting plane (Ψ_T, Δ_T) by deciding $\boldsymbol{\pi}_T$ using (5.60).
  Update H, i.e. adding the new row and column corresponding to Ψ_T.
  Solve the reduced dual problem (5.61).
  Update $\mathbf{w}$ by (5.62).
end for
return $\mathbf{w}$.

for all t. For linear SVM, adding the cutting plane and updating H can be solved in $O(M)$ and $O(T)$ time, respectively, where M is the number of training instances and T is the number of existing cutting planes, and we assume $\mathbf{w}^T\mathbf{x}$ can be evaluated in $O(1)$. However, for kernel SVM, since the feature mapping $\phi(\mathbf{x})$ is not given, we need represent the cutting plane Ψ_t and the model parameter $\mathbf{w}$ as follows:

$$\Psi_t = \sum_{i=1}^{M} \pi_{ti} y_i \phi(\mathbf{x}_i), \tag{5.64}$$

$$\mathbf{w} = \sum_{t=1}^{T} \alpha_t \Psi_t = \sum_{i=1}^{M} \tilde{\alpha}_i \phi(\mathbf{x}_i). \tag{5.65}$$

Therefore the computation cost for finding the cutting plane (i.e. evaluating $f(\mathbf{x}_i)$ for all i) becomes $O(M^2)$, and for updating H becomes $O(T + M^2)$, also assuming $k(\mathbf{x}_i, \mathbf{x}_j)$ can be evaluated in $O(1)$. To reduce this cost, instead of representing Ψ_t as the linear combination of $\{\phi(\mathbf{x}_i)\}$, we seek a small set of basis vectors $\{\mathbf{b}_1, ..., \mathbf{b}_K\}$ where $K \ll M$, such that:

$$\mathcal{W}' = span(\phi(\mathbf{b}_1), ..., \phi(\mathbf{b}_K)) \approx \mathcal{W}, \tag{5.66}$$

$$\Psi_t \approx \hat{\Psi}_t = \sum_{k=1}^{K} \beta_{t,k} \phi(\mathbf{b}_k), \tag{5.67}$$

$$\mathbf{w} \approx \sum_{t=1}^{T} \alpha_t \hat{\Psi}_t = \sum_{k=1}^{K} \alpha_k^* \phi(\mathbf{b}_k). \tag{5.68}$$

Then adding the cutting plane cost $O(MK)$, and updating H takes $O(T + K^2)$.

Two problems need to be solved for this purpose. First, to approximate a cutting

plane Ψ by its projection $\hat{\Psi}$ onto $\mathcal{W}'$, we need to solve the following problem:

$$\min_{\boldsymbol{\beta}} \left\| \Psi - \sum_{j=1}^{K} \beta_j \phi\left(\mathbf{b}_j\right) \right\|^2. \tag{5.69}$$

The solution is $\boldsymbol{\beta} = G^{-1} K_{BX} \hat{\mathbf{y}}$, where G is a $K \times K$ matrix, $G_{ij} = k\left(\mathbf{b}_i, \mathbf{b}_j\right)$; K_{BX} is a $K \times M$ matrix, $K_{BX,ij} = k\left(\mathbf{b}_i, \mathbf{x}_j\right)$; and $\hat{\mathbf{y}}$ is a $M \times 1$ vector, $\hat{y}_i = \pi_i y_i$.

Second, we need to construct the basis set $\{\mathbf{b}_1, ..., \mathbf{b}_K\}$. Ideally we should solve the following problem:

$$\min_{\{\boldsymbol{\beta}_t\}, \{\mathbf{b}_j\}} \sum_{t=1}^{T} \left\| \Psi_t - \sum_{j=1}^{K} \beta_{tj} \phi\left(\mathbf{b}_j\right) \right\|^2, \tag{5.70}$$

which is very similar to the kernel PCA (5.17) and the sparse basis set feature subspace (5.39). Instead of solving the problem using batch method, an incremental method is employed. Assume the current basis set is $\{\mathbf{b}_1, ..., \mathbf{b}_K\}$, which can well approximate the existing cutting planes $\{\Psi_1, ..., \Psi_T\}$. Now a new cutting plane Ψ_{T+1} is introduced, and we want to add a new basis such that this new cutting plane $\mathbf{b}_{K+1}$ can also be accurately approximated. Let $\hat{\Psi}_{T+1}$ be the projection of Ψ_{T+1} onto $\{\mathbf{b}_1, ..., \mathbf{b}_K\}$, then we solve the following problem:

$$\left(\beta_{K+1}, \mathbf{b}_{K+1}\right) = \arg\min_{\beta, \mathbf{b}} \left\| \Psi_{T+1} - \hat{\Psi}_{T+1} - \beta \phi\left(\mathbf{b}\right) \right\|^2. \tag{5.71}$$

The problem is equivalent to:

$$\min_{\beta, \mathbf{b}} \quad \beta^2 \phi(\mathbf{b})^T \phi\left(\mathbf{b}\right) - 2\beta \phi(\mathbf{b})^T \left(\Psi_{T+1} - \hat{\Psi}_{T+1} \right). \tag{5.72}$$

Substitute $\beta = \frac{\phi(\mathbf{b})^T \left(\Psi_{T+1} - \hat{\Psi}_{T+1} \right)}{\phi(\mathbf{b})^T \phi(\mathbf{b})}$ into the problem we can obtain:

$$\max_{\mathbf{b}} \frac{\left(\phi(\mathbf{b})^T \left(\Psi_{T+1} - \hat{\Psi}_{T+1} \right) \right)^2}{\phi(\mathbf{b})^T \phi\left(\mathbf{b}\right)}. \tag{5.73}$$

Therefore we are looking for a $\phi\left(\mathbf{b}\right)$ that is maximally correlated with the residual $\Psi_{T+1} - \hat{\Psi}_{T+1}$. Substitute $\Psi_{T+1} = \sum_{i=1}^{M} \pi_{T+1,i} y_i \phi\left(\mathbf{x}_i\right)$ and $\hat{\Psi}_{T+1} = \sum_{j=1}^{K} \beta_{T+1,j} \phi\left(\mathbf{b}_j\right)$

into the problem, we can get:

$$\max_{\mathbf{b}} \frac{\left(\sum_{i=1}^{M} \pi_{T+1,i} y_i k\left(\mathbf{b}, \mathbf{x}_i\right) - \sum_{j=1}^{K} \beta_j k\left(\mathbf{b}, \mathbf{b}_j\right) \right)^2}{k\left(\mathbf{b}, \mathbf{b}\right)}. \tag{5.74}$$

The problem does not have a closed form solution. We may test all choices of $\mathbf{b}$, or solve the problem by gradient based methods. In particular, the first basis is selected to maximize $\left| \sum_{i=1}^{M} y_i k\left(\mathbf{b}, \mathbf{x}_i\right) / k\left(\mathbf{b}, \mathbf{b}\right) \right|$, which can be interpreted as finding the middle point between the positive instances and the negative instances. To sum up, the cutting plane training for kernel SVM is given in Algorithm 5.

Algorithm 5 Cutting plane training of kernel SVM

Initialization: $\boldsymbol{\alpha}^* = 0$.
for $T = 1$ to T_{max} **do**
  Add a cutting plane (Ψ_T, Δ_T) by deciding $\boldsymbol{\pi}_T$ using (5.60) and (5.68).
  Add a new basis $\mathbf{b}_T$ by solving (5.74).
  Approximate all the cutting planes $\{\Psi_t\}$ by projecting to the basis set $\{\mathbf{b}_t\}$, obtain the projections $\{\boldsymbol{\beta}_t\}$.
  Update H according to $\left\{\hat{\Psi}_t\right\}$.
  Solve the reduced problem (5.61) to obtain weights of the cutting planes $\boldsymbol{\alpha}$.
  Update $\boldsymbol{\alpha}^*$ according to (5.67) and (5.68).
end for
return $\{\mathbf{b}_t\}$ and $\boldsymbol{\alpha}^*$.

5.3 Multiple kernel and multiple instance similarity features

By introducing a feature space Φ via the kernel function $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$, the linear classifiers in Φ are able to capture complex separating boundaries in the original feature space $\mathbb{X}$. The linear scoring function in Φ is:

$$f\left(\mathbf{x}\right) = \mathbf{w}^T \phi\left(\mathbf{x}\right) = \sum\nolimits_{i=1}^{K} \alpha_i k\left(\mathbf{x}, \mathbf{b}_i\right), \tag{5.75}$$

and $\{\phi\left(\mathbf{b}_i\right)\}$ is the set of basis that spans a subspace of Φ in which the normal of the separating hyperplane lies. The kernel functions $\{k\left(\mathbf{x}, \mathbf{b}_i\right)\}$ essentially measure the similarity between the test instance $\mathbf{x}$ and a number of exemplars $\{\mathbf{b}_i\}$. Considering

the similarity values as spanning a new feature space, the scoring function $f(\mathbf{x})$ is also a linear classifier in this similarity feature space. In this section, we generalize this idea by considering using multiple similarity functions, which resembles the multiple kernel learning approach, and using spatial pooling of the similarity values for the visual object detection problem, which essentially implements the idea of multiple-instance learning. We learn the classifier by directly max-margin in the similarity feature space, and use feature selection and coarse-to-fine scheme to effectively find a small number of exemplars to improve the efficiency of the classifier.

5.3.1 MKMIS features

Similarity features

Given a sample instance represented by a low-level feature vector $\mathbf{x}$, and given a set of exemplar instances $E = \{\mathbf{x}_{e1}, \mathbf{x}_{e2}, ..., \mathbf{x}_{eM}\}$, we can derive a new feature space by using the similarity between $\mathbf{x}$ and each exemplar in E as features:

$$\mathbf{s} = \left[S(\mathbf{x}_{e1}, \mathbf{x}), ..., S(\mathbf{x}_{eM}, \mathbf{x})\right]^T, \tag{5.76}$$

where S is the similarity function. We learn a classifier using the similarity feature vector $\mathbf{s}$, rather than the low-level feature vector $\mathbf{x}$. A linear scoring function using the similarity feature is similar to a kernel machine using the low-level features, both have the form $f(\mathbf{x}) = f\left(\{S(\mathbf{x}, \mathbf{x}_j)\}_{j=1}^{M}\right)$, while the kernel trick requires the similarity function to be a valid Mercer kernel $S(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$, such that $k(\mathbf{x}_i, \mathbf{x}_j)$ is the inner product in some transformed feature space $\phi(\mathbf{x})$, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. However, this requirement is relieved when the similarity is simply considered as spanning a new feature space, and the classifier we have learned shows different properties from those methods using the kernel trick.

Multiple kernel similarity

There are many possible methods for measuring the similarity between two instances. The similarity can be measured according to information from various sources, using low-level features of different natures. Different similarity functions can be employed to adapt to the geometry of the distribution in the low-level feature space, as in [114]. For example, the similarity function can be calculated based on the L-1 or L-2 distance,

the Mahalanobis distance with various covariance matrices, or by using the RBF kernel with various bandwidths. The multiple kernel learning method aims to learn a kernel k by combining a number of base kernels $\{k_p\}_{p=1}^{P}$:

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = g\left(\left\{k_p\left(\mathbf{x}_i, \mathbf{x}_j\right)\right\}_{p=1}^{P}\right), \tag{5.77}$$

and then k is plugged into kernel methods, e.g. a kernel SVM.

For the similarity features, we concatenate the similarity values computed using various methods into one vector:

$$S\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left[S_1\left(\mathbf{x}_i, \mathbf{x}_j\right), ..., S_P\left(\mathbf{x}_i, \mathbf{x}_j\right)\right]^T. \tag{5.78}$$

Compared with using a combined kernel, more information is preserved in the concatenated similarity feature vector. Consider that each instance is represented by a number of feature channels $\mathbf{x} = \left[\mathbf{x}^1, ..., \mathbf{x}^P\right]$, and a similarity function is defined for each feature channel: $S_p\left(\mathbf{x}_i, \mathbf{x}_j\right) = S_p\left(\mathbf{x}_i^p, \mathbf{x}_j^p\right)$. Due to the existence of noisy feature channels, two positive instances may not be similar holistically, but highly similar in some feature channels, while the similar feature channels for different pairs of instances may vary. Therefore, a similarity vector obtained by concatenation preserves more information than is possible by combining the similarity values into one single scalar value.

Multiple instance similarity

In visual object detection, the objects are usually not perfectly aligned in training or testing. This could be due to inaccuracy in the labeled training set or deformation of the object, e.g. the articulated motion of human body parts. Therefore, the best match of a rectangular region from an exemplar bounding box may be at a shifted position in another object instance's bounding box. For example, for different pedestrians, the heads can be at slightly different locations in different bounding boxes, although the pedestrians have been roughly aligned. To take this into account, we introduce the notion of multi-instances into the similarity features.

We divide the object's bounding box into a number of rectangular regions, and use each region as a feature channel in the multi-kernel similarity feature. Denote the location of an exemplar region as the anchor, and denote the set of rectangular regions in a neighborhood of the anchor as the support set. The support set corresponds to the bag of instances in the multiple-instance learning literature. Due to the alignment inaccu-

racy, the best match can be any item in the support set. Ideally, a positive exemplar should have high similarity to at least one item in a positive support set (i.e. the support set from a positive instance), and low similarity to all items in a negative support set. This is essentially the basic assumption of multiple instance learning (MIL), which states that a positive bag contains at least one positive instance, while a negative bag does not contain any positive instances. Therefore the multiple-instance learning can be implemented at the feature level by simply taking the maximum similarity value to the support set:

$$\hat{S}\left(\mathbf{x}_i^p, \mathbf{x}_j^p\right) = \max_{s \in \mathcal{S}_p} \left\{ S\left(\mathbf{x}_i^p, \mathbf{x}_{j,s}^p\right) \right\}, \tag{5.79}$$

where $\mathcal{S}_p$ is the support set for the anchor p.

Note that the above formulation is not symmetric, i.e. $\hat{S}\left(\mathbf{x}_i^p, \mathbf{x}_j^p\right) \neq \hat{S}\left(\mathbf{x}_j^p, \mathbf{x}_i^p\right)$. If the similarity function is positive semi-definite, a matching kernel can be made as:

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{1}{2} \sum_{p=1}^{P} \left(\hat{S}\left(\mathbf{x}_i^p, \mathbf{x}_j^p\right) + \hat{S}\left(\mathbf{x}_j^p, \mathbf{x}_i^p\right) \right). \tag{5.80}$$

This kernel is positive semi-definite, and ignores the misalignment error as long as it can be covered in the support set. Therefore it can be plugged into any kernel machine from the textbook, e.g. used in a kernel SVM. In addition, the cost of location displacement can be taken into account, e.g. as done in the graph-matching kernel [115].

Discussions

Capturing the non-linear separating boundaries and multi-modal distributions in the original featurespace $\mathbb{X}$ are two main benefit of the kernel methods, by introducing a non-linear feature map $\phi\left(\mathbf{x}\right)$. Another most intuitive and most widely used technique for this purpose is "divide and conquer", where the object class is divided into sub-categories, either manually [116] or automatically [90, 117], using hard assignment or soft assignment. Then models are learned individually for each sub-category such that, together, they cover the whole range of object appearance. A test instance is classified as belonging to the object class if it belongs to any of the sub-categories, or by considering the sub-category scores as features and applying a high-level model. This technique introduces issues such as deciding the number and range of the sub-categories, which can be quite ambiguous. For example, sub-categories can be defined for each part of the object, but hard to define for the object holistically, due to com-

binatorial explosion. The recent deformable part model [80] and grammar model [81] are designed to capture the part-full structure, and to model the sub-categorization for the full object as well as for the parts. A rich set of grammar is included to describe the object category, and the geometric configurations of the parts can be well modeled. However, the grammar model is hard to train due to the large number of tunable parameters, and is an overkill for basic tasks such as object detection. Furthermore, as shown in a recent study [118], the part detector is still the weakest link in the grammar model.

Instead of explicit sub-categorization, the similarity feature performs implicit sub-categorization and thus is able to capture the multi-modal distribution. Compared with the low-level features extracted directly from the sensor data, the similarity feature is a middle-level feature that possesses semantic meanings. A classifier trained using the similarity features is similar to the high-level model trained on the sub-category scores, assuming that each exemplar is a sub-category classifier and the similarity value is the classifier's score. Furthermore, the use of multiple local exemplars performs the sub-categorization at the part level, and the classifier is trained to favor some combination while disfavor some others, therefore avoiding directly handling the combinatorial explosion due to explicitly modeling all sub-categories for the full object.

The similarity feature is closely related to the nearest neighbor methods. The nearest neighbor methods usually need a proper similarity metric for dealing with high-dimensional feature space and for combining information from heterogeneous sources. For the kernel methods, e.g. kernel SVM, the kernel indeed measures the similarity between examples. A good kernel function is essential; this is studied within the scope of learning kernels and multiple-kernel learning (MKL) [55], where the kernel is obtained by combining a number of base kernels, which differ either in their features, their functional form, or just their parameters. Therefore it is natural to consider multiple kernels in the similarity feature. In our design, each feature channel has a small number of dimensions such that simple similarity metrics work well. For example, a feature channel only represents a small rectangular region inside the object bounding box. Instead of forming a combined kernel as in many MKL methods, all the similarity values calculated using various features or similarity functions are concatenated to form a single feature vector, which is then used with learning algorithms that performs effective feature selection. Thus, the sub-categorization for each part can be learned separately, and the part-full structure of the object can be modeled, since the full score is obtained by summarizing the part scores.

5.3.2 Learning with MKMIS features

Formulation of the learning problem

We learn the classifier by structural risk minimization, directly using the similarity features:

$$\min \quad \lambda R\left(f\right)+L\left(f\right), \tag{5.81}$$

where $R\left(f\right)$ is the regularization term that constrains the VC dimension of the classifier, while $L\left(f\right)$ is the empirical loss in the training set. The learnability is guaranteed by the statistical learning theory, which states that the generalization error is probabilistically bounded by a function of the VC dimension and the empirical loss. We focus on linear models $f\left(\mathbf{s}\right)=\boldsymbol{\alpha}^{T}\mathbf{s}$, and in particular we consider the L-2 regularized L-1 loss SVM:

$$\min_{\boldsymbol{\alpha}} \quad \tfrac{\lambda}{2}\boldsymbol{\alpha}^{T}\boldsymbol{\alpha}+\sum_{i=1}^{N}\max\left(0,1-y_{i}\boldsymbol{\alpha}^{T}\mathbf{s}_{i}\right), \tag{5.82}$$

where N is the number of training instances. The regularization term $R\left(f\right)=R\left(\boldsymbol{\alpha}\right)=\frac{1}{2}\boldsymbol{\alpha}^{T}\boldsymbol{\alpha}$ encourages the classifier with a large margin in the similarity feature space, and the hinge loss $L\left(f\right)=L\left(\boldsymbol{\alpha}\right)=\sum_{i=1}^{N}\max\left(0,1-y_{i}\boldsymbol{\alpha}^{T}\mathbf{s}_{i}\right)$ relaxes the classifier to deal with linearly non-separable training data.

Notice that the learning problem (5.82) differs from that of learning a SVM with reduced basis set (5.83) only in the regularization term:

$$\min_{\boldsymbol{\alpha}} \quad \tfrac{\lambda}{2}\boldsymbol{\alpha}^{T}G\boldsymbol{\alpha}+\sum_{i=1}^{N}\max\left(0,1-y_{i}\boldsymbol{\alpha}^{T}\mathbf{s}_{i}\right), \tag{5.83}$$

assuming $\mathbf{s}_{i}=\left[k\left(\mathbf{x}_{i},\mathbf{x}_{e1}\right),...,k\left(\mathbf{x}_{i},\mathbf{x}_{eM}\right)\right]$, and G is a $M\times M$ matrix, $G_{ij}=k\left(\mathbf{x}_{ei},\mathbf{x}_{ej}\right)$. However, the formulation (5.83) is not applicable if the similarity feature is not a proper kernel, e.g. for the multiple-instance similarity features.

The multi-kernel similarity feature results in a linear scoring function as follows:

$$f\left(\mathbf{x}\right)=\sum_{j\in\{1,...,M\},p\in\{1,...,P\}}\alpha_{j,p}S_{p}\left(\mathbf{x},\mathbf{x}_{ej}\right), \tag{5.84}$$

where $\{\alpha_{j,p}\}$ can be learned directly from the linear SVM. By comparison, current works on MKL focus on learning a combined kernel $k\left(\mathbf{x}_{i},\mathbf{x}_{j}\right)=\sum_{p=1}^{P}\mu_{p}k_{p}\left(\mathbf{x}_{i},\mathbf{x}_{j}\right)$

that is applied to all support vectors, and the classifier is:

$$f(\mathbf{x}) = \sum_{j=1}^{M} \alpha_j \sum_{p=1}^{P} \mu_p k_p(\mathbf{x}, \mathbf{x}_{ej}) \tag{5.85}$$

If we treat the combined kernel as a similarity metric, the MKL learns a global similarity metric, which is applied to every exemplar. In contrast, our method learns exemplar-specific similarity metrics:

$$S_{ej}(\mathbf{x}, \mathbf{x}_{ej}) = \sum_{p=1}^{P} \alpha_{j,p} S_p(\mathbf{x}, \mathbf{x}_{ej}). \tag{5.86}$$

Therefore, our model contains more parameters than MKL ($M \times P$ vs. $M + P$), and is able to explore a richer set of functions. However, the large number of parameters imposes learning difficulties, which we shall discuss in the next part.

Learning an efficient classifier

We want to learn a parsimonious classifier such that only a small sub-set of $\{\alpha_{j,p}\}$ is non-zero. The forward feature-selection scheme from section 2.4 is a suitable choice for this purpose, which sequentially selects the exemplars to optimize the objective of (5.82). However, training is still computationally demanding since, we have a huge number of similarity features ($M \times P$), which requires massive storage and computational cost. All of these features are tested so as to select the best feature. To relieve this burden, we introduce a coarse-to-fine scheme that consists of two main points: First, instead of using the full exemplar set, we use a reduced sub-set. To do this effectively, we may take a random sample of the exemplars, or learn a codebook by clustering. In practice, we learn a codebook for each feature channel separately. Second, to minimize the performance degradation due to using a small exemplar set, we optimize the exemplars themselves after selecting the similarity features, by solving the following optimization problem:

$$\min_{\{\mathbf{x}_{ej}\}} \sum_{i=1}^{N} L\left(y_i, \sum_{j=1}^{M} w_j S(\mathbf{x}_i, \mathbf{x}_{ej})\right). \tag{5.87}$$

The regularization term $R(\mathbf{w})$ does not need to be considered since we use L-2 regularization which is independent of the exemplars. The loss function is usually non-convex with respect to the exemplars, and could be non-differentiable. We use the

coordinate sub-gradient descent method to sequentially optimize the exemplars $\{\mathbf{x}_{ej}\}$. The gradient/sub-gradient is:

$$\sum_{i=1}^{N} \frac{\partial L}{\partial f_i} \frac{\partial f_i}{\partial s_{ij}} \frac{\partial s_{ij}}{\partial \mathbf{x}_{ej}}. \tag{5.88}$$

We take the hinge loss and the Gaussian similarity function $s_{ij} = \exp\left(-\gamma \left\|\mathbf{x}_i - \mathbf{x}_{ej}\right\|_2^2\right)$ for example. A sub-gradient for the hinge loss can be calculated according to follows:

$$\begin{aligned} \partial L/\partial f_i &= -y_i I\left(y_i f_i < 1\right), \\ \partial f_i/\partial s_{ij} &= w_j, \\ \partial s_{ij}/\partial \mathbf{x}_{ej} &= -2\gamma s_{ij}\left(\mathbf{x}_i - \mathbf{x}_{ej}\right), \end{aligned} \tag{5.89}$$

where $I\left(y_i f_i < 1\right)$ is an indicator function, $I = 1$ if $y_i f_i < 1$, otherwise $I = 0$. Therefore, the effect is that the exemplar $\mathbf{x}_{ej}$ will only be affected by the in-margin training instances. If w_j is positive, $\mathbf{x}_{ej}$ will be pulled towards the positive in-margin instances while being pushed away from the negative in-margin instances, and vice versa. The strength with which a training instance pushes/pulls an exemplar is proportional to the similarity s_{ij}, as well as the distance $\mathbf{x}_i - \mathbf{x}_{ej}$, such that those very distant training instances will have a small influence due to the fast decay of s_{ij}. The empirical loss is always improved by modifying the exemplars, while the regularization term is not affected. Therefore, we expect that the generalization performance can be improved, especially when a very small sub-set of exemplars is used. This exemplar refinement is also adopted in other algorithms, e.g. the learning vector quantization (LVQ) [119] for nearest-neighbor classifiers, and the exhaustive search method is used in [120].

5.3.3 Experimental Evaluation

In this section we demonstrate our approach on synthetic data and real image data. First, we study the properties of our approach using synthetic Gaussian mixture data, showing that how the similarity feature, the forward feature selection learning, and the coarse-to-fine learning together contribute to an efficient and effective algorithm. Then, we test with real-world image data to further address the benefit of multiple-kernel and multiple-instance similarity features.

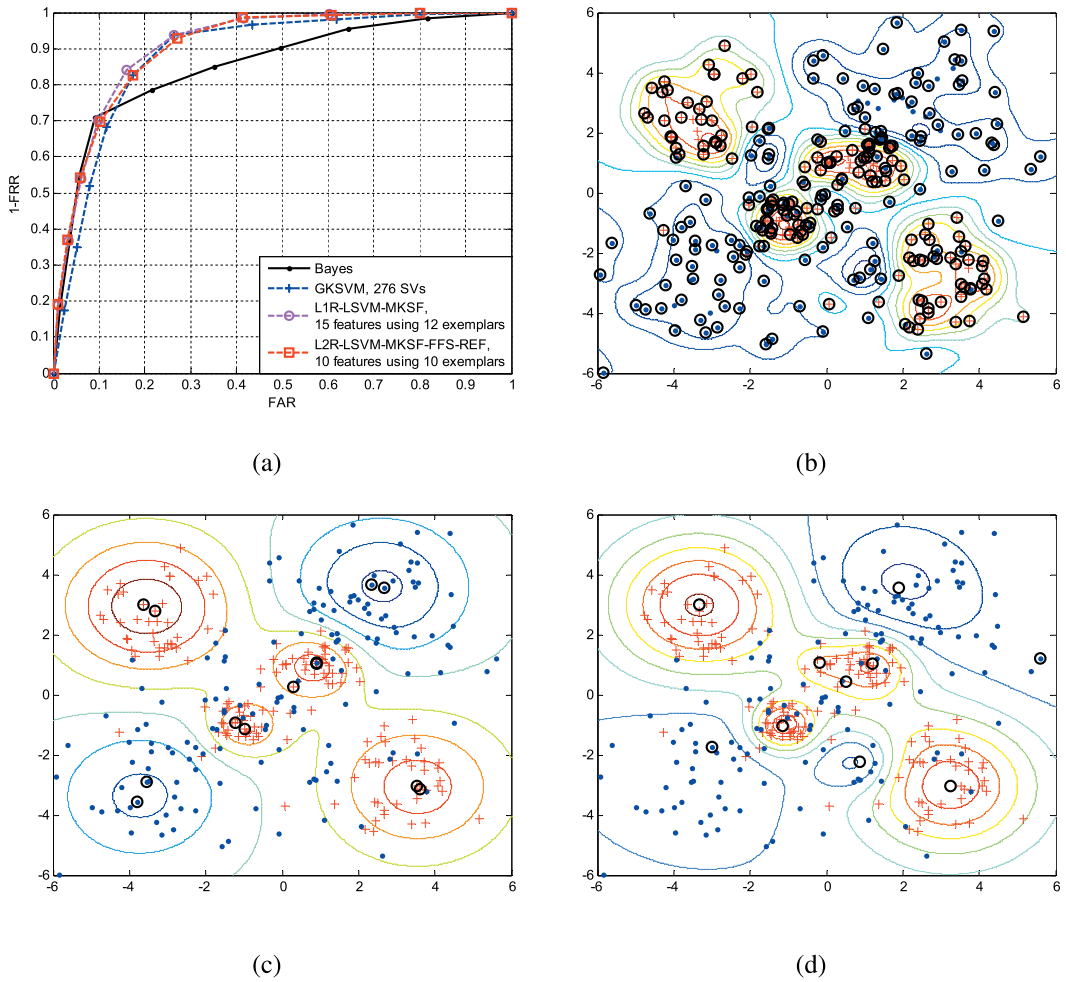
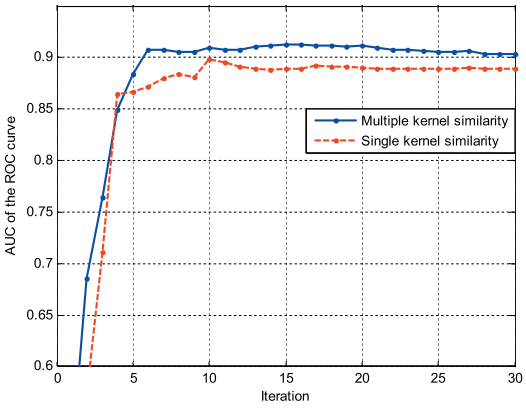


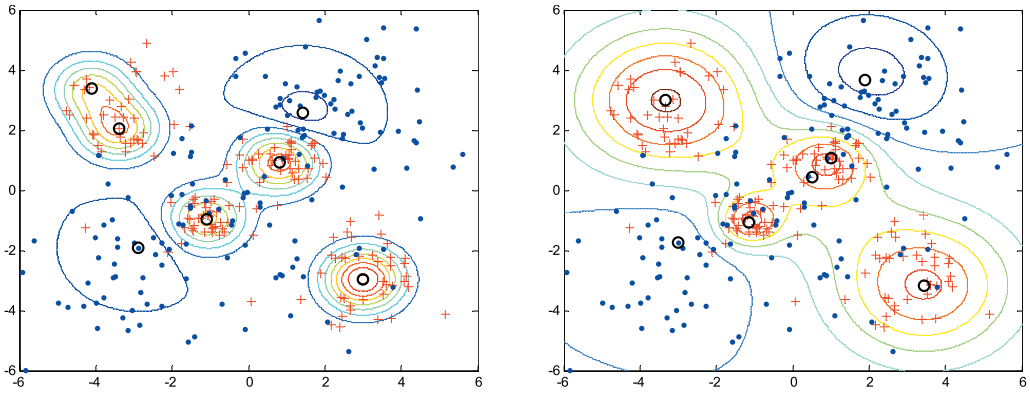
Figure 5.7: (a) The ROC curve of various methods. (b) Score contours of GKSVM. (c) Score contours of L1R-LSVM-MKSF. (d) Score contours of L2R-LSVM-MKSF-FFS-REF. All training instances are used as exemplars for the similarity feature. The selected support vectors/exemplars are shown as black circles. For the L2R-LSVM-MKSF-FFS-REF method, the final exemplars may not be in the initial exemplar set, due to the refinement stage.

Synthetic toy data

We generate a 2-D Gaussian mixture distribution of points for both the positive and the negative dataset. The positive class has four components with equal weights: $\{\mu_{+1} = (3,-3), \sigma_{+1} = 1\}$, $\{\mu_{+2} = (-3,3), \sigma_{+2} = 1\}$, $\{\mu_{+3} = (1,1), \sigma_{+3} = 0.5\}$, $\{\mu_{+4} = (-1,-1), \sigma_{+4} = 0.5\}$. The negative class has two components with equal weights: $\{\mu_{-1} = (2,2), \sigma_{-1} = 2\}$ and $\{\mu_{-2} = (-2,-2), \sigma_{-2} = 2\}$. Therefore, the positive class forms two compact clusters and two less compact clusters, while the distribution



(a)

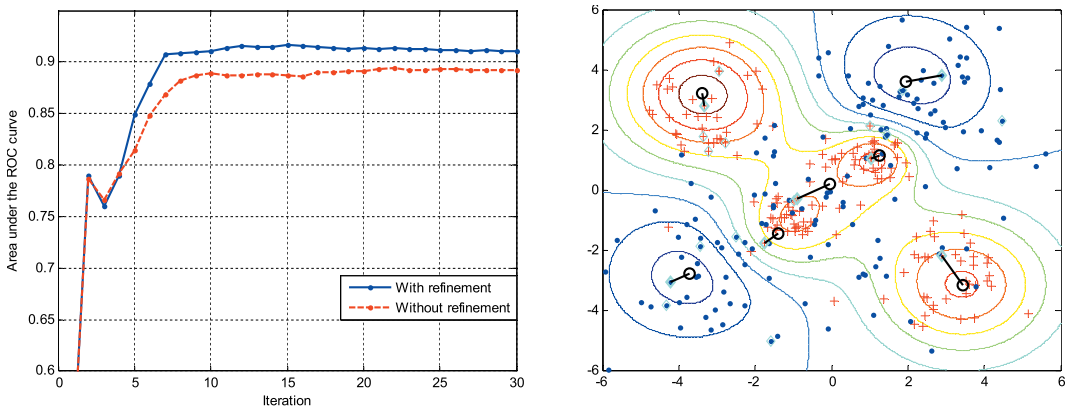


(b) (c)

Figure 5.8: Comparison of single-kernel and multiple-kernel similarity features. (a) AUC of L2R-LSVM-SF-FFS vs. L2R-LSVM-MKSF-FFS. (b) Score contours of L2R-LSVM-SF-FFS, using 7 similarity features. (c) Score contours of L2R-LSVM-MKSF-FFS, using 7 similarity features.

of the negative class is rather flat. The training set consists of 160 points for each class, and the test set consists of 800 points for each class.

In Fig. 5.7, we compare the holistic Gaussian kernel SVM (GKSVM) and the multi-kernel similarity features (MKSF) with two different feature-selection methods, i.e. the L-1 regularized linear SVM (L1R-LSVM) and the L-2 regularized linear SVM (L2R-LSVM) with our forward feature selection (FS) method. The refinement stage (REF) is also implemented for L2R-LSVM, although we find that the benefit of this diminishes when the codebook is large or when many features are selected. For GKSVM, the kernel bandwidth γ and the trade-off parameter λ are selected by cross validation. 276 SVs are selected by GKSVM. All training instances are included in the exemplar set, and the multi-kernel similarity feature is calculated using Gaus-



(a) (b)

Figure 5.9: Effect of exemplar refinement using a small exemplar set. (a) AUC of L2R-LSVM-MKSF-FFS vs. L2R-LSVM-MKSF-FFS-REF. (b) Score contours of L2R-LSVM-MKSF-FFS-REF using a small number of exemplars. The original exemplars are shown as cyan diamonds, and the refined exemplars are shown as black circles.

sian kernels, with $\gamma \in \{0.2,\ 0.5,\ 0.7, 1.0, 1.5, 2.0\}$, resulting in 6 similarity features for each exemplar. For L1R-LSVM, the number of selected features is controlled by varying λ. 15 similarity features using 12 exemplars are selected; further sparsity will cause significant performance loss. For L2R-LSVM, the number of selected features is explicitly specified, and we show the result of using 10 features. As Fig. 5.7a shows, a small number of similarity features is sufficient to outperform the single kernel GKSVM using a large number of support vectors. The GKSVM and the LSVM using similarity features favor different sets of exemplars, as shown in Figs. 5.7b to 5.7d. GKSVM selects the support vectors, i.e. all training instances inside the margin, while LSVM first selects the exemplars near the cluster centers, and then selects more exemplars to refine the decision boundary.

We illustrate how the multiple kernels improve over the single-kernel similarity features (SKSF). In Fig. 5.8, we compare the best SKSF using $\gamma = 1.0$ and the MKSF. We show the area under the ROC curve (AUC) as a function of the number of selected features, which is an indicator of the overall performance of the classifier. The plot shows that MKSF consistently outperforms SKSF, and very good results can be achieved by using as few as 6 features in MKSF. Figs. 5.8b and 5.8c show the contours of the scoring function when 7 features are selected. It is clearly observable that, using MKSF, basis functions of various bandwidths are selected to approximate the underlying distribution.

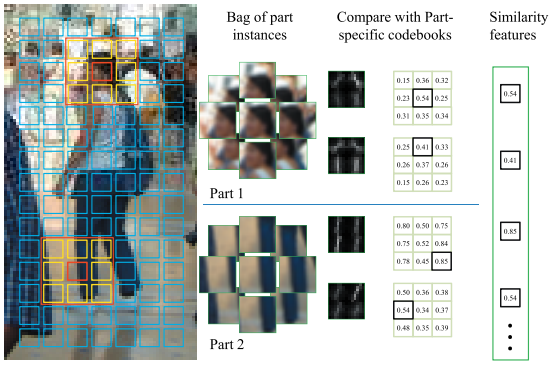


Figure 5.10: Illustration of the feature extraction scheme of the multi-kernel, multi-instance similarity feature for the INRIA pedestrian dataset.

Finally we show that the refinement stage is essential when training with a small exemplar codebook. A small codebook is used by randomly picking 20 of the 320 training instances as exemplars. Fig. 5.9a shows the area under the ROC curve as a function of selected features. The refinement consistently improves the performance. The contour of the scoring function is shown in Fig. 5.9b, along with the exemplar locations before and after refinement. It can be observed that the exemplar positions are pulled towards the cluster centers, and thus the basis functions are better aligned with the training data.

Pedestrian detection

In this part we demonstrate the multi-kernel multi-instance similarity feature on the INRIA pedestrian dataset. The linear SVM with the HOG feature [19] is known to perform very well on this dataset, and the idea of learning a holistic template by linear SVM has been well received in the field. Although non-linear kernel machines usually achieve a better performance in classifying objects with highly variable appearance, they are rarely used in practical systems, except when the scoring function can be efficiently evaluated—for example, approximated by linear functions of the image descriptors [63, 64]. In the following work, we simplify a non-linear kernel machine in another manner, treating it as a linear machine using the similarity features; and we train an efficient classifier with a small number of similarity features, using our proposed feature selection scheme.

The HOG feature is used as the low-level image descriptors. As in [19], 16×16 HOG blocks are taken from a regular grid with an overlap of 8 pixels between adjacent blocks. The HOG histogram from a group of four 8×8 cells in the block

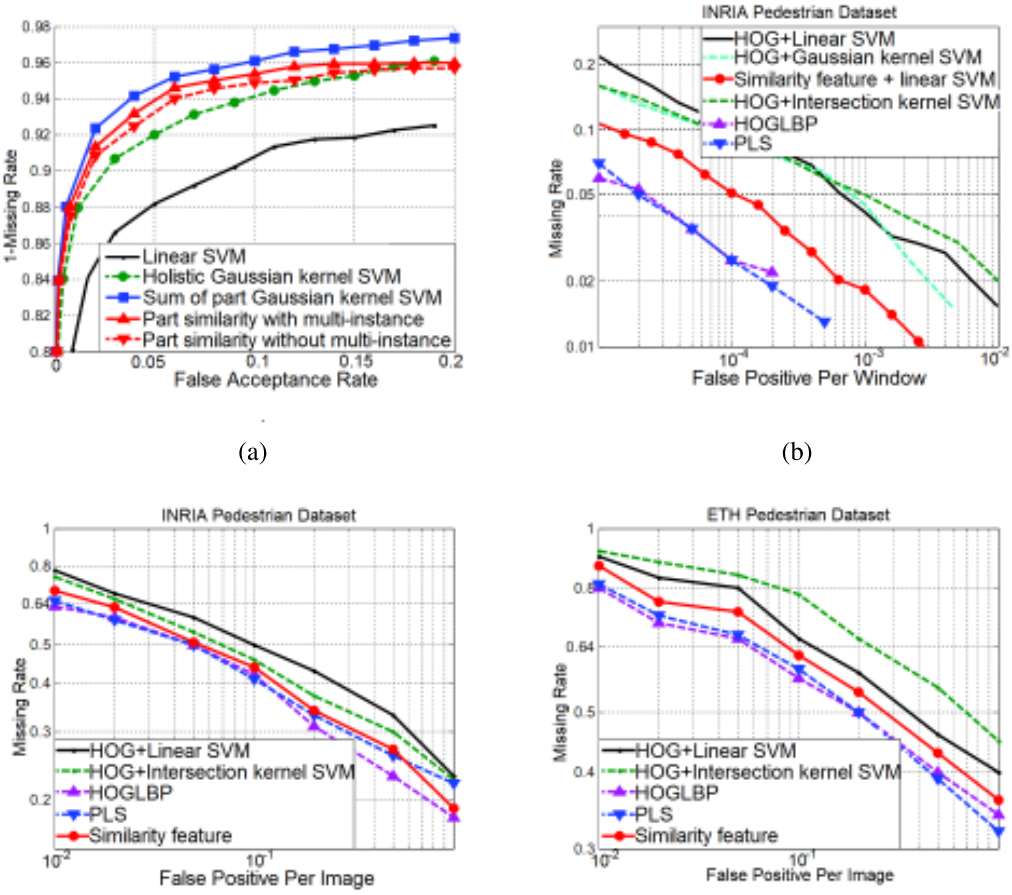


Figure 5.11: Results on the pedestrian detection problems. (a) The ROC curves on the dataset with 40,000 hard negative instances. (b) FPPW on the full INRIA test dataset. (c) FPPI on the full INRIA test dataset. (d) FPPI on the full ETH test dataset.

are concatenated together, and L-2 normalization is applied. No spatial interpolation, orientation interpolation, or dimensionality reduction [80] is performed, such that the low-level image descriptors can be calculated very fast using the integral image. The object bounding box size in 128×64, hence there are 105 HOG blocks. Each HOG block corresponds to a feature channel in our similarity feature. For multiple-instance matching, a bag of 9 instances is generated by displacing the anchor site by [-4, 0, 4] pixels in the x and y directions. A codebook of size 400 is learned for each feature channel using k-means clustering and using only the positive dataset. Therefore, the full similarity feature set has 42,000 features, which is already significantly reduced compared to using the full set of part exemplars, i.e. $105 \times 2,474 = 259,770$ similarity features. The feature extraction scheme is illustrated in Fig. 5.10

The training set contains 2,474 positive examples, and the test set contains 1,178 positive examples. For the negative examples, we collect a hard negative dataset of 40,000 examples from the hard negative examples (score > -1) of a linear SVM + HOG detector, and use half for training and half for testing.

First, we show the classification performance on this dataset in Fig. 5.11a. We compare the performances of the holistic linear SVM, the holistic Gaussian-kernel SVM with kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$, the sum-of-part-Gaussian-kernel SVM with kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{p=1}^{P} \exp(-\gamma\|\mathbf{x}_i^p - \mathbf{x}_j^p\|^2)$, and linear SVM with the multi-kernel, multi-instance similarity features, selecting 1,200 features. The holistic Gaussian-kernel SVM uses 7,037 support vectors, and achieves a recall rate of about 92% recall when the false acceptance rate (FAR) is 5%, which is about a 4% improvement over linear SVM. The performance is further improved by the sum-of-part-Gaussian-kernel SVM, which uses 5,581 support vectors and achieves a recall rate of about 95% when FAR=5%. However, both the holistic Gaussian-kernel SVM and the sum-of-part-Gaussian-kernel SVM are expensive to evaluate. Our similarity feature linear SVM uses only 1,200 similarity features, and achieves a slightly worse performance than the sum-of-part-Gaussian-kernel SVM, about a 94% recall rate when FAR=5%. Essentially we can write the sum-of-part-Gaussian-kernel SVM in a linear form, using $5,581 \times 105 = 586,605$ part-similarity features, so our proposed scheme significantly reduces the classifier's complexity. We have also shown the result of part-similarity features without using the multi-instance evaluation, and the performance appears to be degraded.

In Fig. 5.11b and Fig. 5.11c, we show the performance evaluated on the full INRIA dataset, and, in Fig. 5.11d the ETH dataset. In Fig. 5.11b, we compare our method with the linear SVM, the Gaussian-kernel SVM, and the intersection-kernel SVM [63] (using the corrected results from [121]). The sum-of-part-Gaussian kernel SVM is too time-consuming to evaluate, therefore it is not included in this section. We have also shown the performance of two other state-of-the-art methods using more powerful low-level features, i.e. the HOG-LBP [122] with a linear SVM and the PLS method [42] with a large set of heterogeneous low-level features. The missing rate–FPPW curves and the missing rate–FPPI curves are provided for the INRIA dataset in Fig. 5.11b and Fig. 5.11c, respectively, and for the ETH dataset in Fig. 5.11d. For the INRIA dataset, using the FPPW evaluation metric, our approach reduces the false positive rate by about an order of magnitude (e.g. from 10^{-4} to 10^{-5} when the missing rate = 0.1 in Fig. 5.11b), compared to the HOG + linear SVM approach. Our approach also outperforms two non-linear kernel SVMs operating on holistic HOG

feature representation, indicating the benefit of using part similarities over holistic similarities. However, the performance gain does not come without a cost. While our approach is much more efficient than the Gaussian-kernel SVM due to the small number of part similarity features, a considerable number (1,200 in our experiment) of part exemplars are still used, which involves about 10 times more computational cost than either the linear SVM with HOG (which can be considered as using 105 part exemplars) or the intersection-kernel SVM with HOG. However, we note that the testing efficiency can be effectively improved by building cascade detectors. Besides, it is evident that the low-level features have a significantly impact on the performance, as demonstrated by the HOG-LBP and the PLS approach. We believe that our method will also benefit from the introduction of heterogeneous descriptors, which we leave for our future work.

5.3.4 Conclusions

In this chapter, we have proposed a middle-level feature for visual object detection, and have designed a learning framework to obtain efficient classifiers using this feature. The middle-level feature is based on the similarity to exemplars, and elements of multiple-kernel learning and multiple-instance learning are incorporated at the feature level, enhancing the similarity feature in several ways. First, more complex decision boundaries can be efficiently described using a small number of similarity measures. Second, the part-full structure of visual objects as well as the geometric deformations can be captured at the feature level, rather than resorting to complicated model design or learning methods. Our forward feature-selection scheme and coarse-to-fine learning scheme are tailored for the learning with a high-dimensional data problem when using similarity features; they also produce an efficient classifier by using a small number of features.

Our work can be extended in many ways. Choosing the most appropriate low-level descriptor and similarity metric is an important issue for real-world applications. The refinement procedure can be extended to learn other parameters of the similarity function, for example, kernel bandwidth for the Gaussian kernel. This topic is related to the metric learning problem studied in the nearest-neighbor classifiers. For the classifier structure, a universal codebook can be used for all part locations, such that the effective complexity of the classifier can be reduced by encouraging the re-use of codewords at different part locations. The codebook-based approach is also amenable to multi-class problems, in which codewords can be shared between different classes.

Chapter 6 Conclusions

The emphasis of this thesis is machine learning methods used for training a visual object detector. Though designing features and other issues specific to analyzing the visual information are more familiar to those people from image processing or working on vision applications, the machine learning techniques are the key to making decisions from the collection of information acquired from the sensors, and have far-reaching importance in a wide range of applications beyond visual object detection. Our study is centered around the structural risk minimization principle for supervised learning, for which the foundation is laid in the past several decades by a number of researchers, and which receives popularity in just about the last 10 to 15 years, due to a number of powerful tools being made practical, such as the support vector machines and the Boosting methods.

After a brief review of the structural risk minimization principle and the loss functions for classification in Chapter 2, we have studied several recent algorithms for training the support vector machines. These algorithms exploits the particular formulation of SVM problems. Studying the details of the optimization process grants us better understanding of the learning process, and also motivates the embedded forward feature selection method, which is expressed in the general formulation for many learning problems. The feature selection method demonstrates its power for SVM and logistic regression in our illustrating experiments. The ranking and structural prediction problems are also studied, for they are also useful learning problems that have been successfully applied in visual object detection.

The demand for high efficiency of the detector gives rise to the wide acceptance of Boosting cascade detectors, and the whole Chapter 3 is devoted to this topic. We have studied methods for improving the performance and efficiency of the Boosting algorithm. The classifier performance is improved by solving the totally-corrective AdaBoost problem using continuous-valued weak classifiers derived from the functional optimization formulation of AdaBoost algorithms. Then regarding the training time complexity, we employ the partial least square regression as a feature subset selection method, and train weak classifiers for a subset of features rather than for all the features. To build the cascade, three topics are discussed, i.e. training with asymmetric objectives, selecting the operating point for each stage, and recycle existing informa-

tion for better efficiency. For learning with the asymmetric objective, typical methods include cost-sensitive learning and direct optimization of the Recall for a fixed FAR value (e.g. 0.5). For selecting the operating point, the one-point and two-point planning methods both improves efficiency by using less weak classifiers and rejecting more negative windows in the early stages. Re-use existing weak classifiers and features further improves the efficiency of the cascade, and our biased selection strategy gives better performance than existing recycling approaches.

Lastly we have studied the kernel methods, which, although very powerful, is rarely used in existing visual object detection systems for its high computational cost using non-linear kernels. Therefore methods for accelerating the kernel SVM are discussed, including approximating the scoring function, approximating the feature map, and selecting a reduced set of basis vectors in the learning process. Inspired by the kernel methods, we consider using the similarity as features and directly train linear classifiers with the similarity features. The elements of multi-kernel and multi-instance can be introduced for measuring the similarity between visual objects, by applying spatial max-pooling to the similarities with a number of local exemplars. Forward feature selection and exemplar refinement are also applied in learning the classifier, to improve the performance and efficiency of the detector.

To sum up, we have made a number of improvements to existing techniques for learning a visual object detector. These improvements enhance existing methods in both accuracy and efficiency. However, most of our work addresses the fundamental aspects of the learning problems, and our study is focused on using the sliding window scheme for detecting a single class of objects. Future research can be carried out in many directions. For example, it will be interesting to study other properties and problems particular to the visual object detection scenario, which should bring another fold of improvement to the detector performance. High-level analysis can be performed based on the detector's output, and the machine learning techniques are also expected to act an important role in these applications.

Bibliography

[1] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:509–522, 2002.

[2] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[3] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

[4] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.

[5] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(1):39–51, 1998.

[6] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(1):23–38, 1998.

[7] Bastian Leibe, Alan Ettlin, and Bernt Schiele. Learning semantic object parts for object categorization. *Image and Vision Computing*, 26(1):15–26, 2008.

[8] Christian Wojek, Stefan Walk, and Bernt Schiele. Multi-cue onboard pedestrian detection. In *CVPR*, pages 794–801, 2009.

[9] Timo Ojala, Matti Pietikinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, 1996.

[10] Thomas Serre, Lior Wolf, Stanley M. Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):411–426, 2007.

[11] Jim Mutch and David G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision*, 80(1):45–57, 2008.

[12] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[13] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *ICIP (1)*, pages 900–903, 2002.

[14] Genquan Duan, Chang Huang, Haizhou Ai, and Shihong Lao. Boosting associated pairing comparison features for pedestrian detection. In *ICCV Workshop*, 2009.

[15] Chang Huang and Ramakant Nevatia. High performance object detection by collaborative learning of joint ranking of granules features. In *CVPR*, pages 41–48, 2010.

[16] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *CVPR*, 2007.

[17] Thomas Deselaers and Vittorio Ferrari. Global and efficient self-similarity for object classification and detection. In *CVPR*, pages 1633–1640, 2010.

[18] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas S. Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *CVPR*, pages 3360–3367, 2010.

[19] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005.

[20] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cdric Bray. Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision, Prague, 2004*, 2004.

[21] Robert M. Haralick, K. Shanmugam, and Its'hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621, 1973.

[22] Jing Huang, Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih. Image indexing using color correlograms. In *CVPR*, pages 762–768, 1997.

[23] Satoshi Ito and Susumu Kubota. Object classification using heterogeneous co-occurrence features. In *ECCV (5)*, pages 209–222, 2010.

[24] Yi Yang and Shawn Newsam. Spatial pyramid co-occurrence for image classification. In *ICCV*, 2011.

[25] Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *ECCV*, 2006.

[26] Oncel Tuzel, Fatih Porikli, , and Peter Meer. Human detection via classification on riemannian manifolds. In *CVPR*, 2007.

[27] Dariu Gavrila. A bayesian, exemplar-based approach to hierarchical shape matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(8):1408–1421, 2007.

[28] Andreas Opelt, Axel Pinz, and Andrew Zisserman. Learning an alphabet of shape and appearance for multi-class object detection. *International Journal of Computer Vision*, 80(1):16–44, 2008.

[29] Jamie Shotton, Andrew Blake, and Roberto Cipolla. Multiscale categorical object recognition using contour fragments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(7):1270–1281, 2008.

[30] Vittorio Ferrari, L. Fevrier, Frédéric Jurie, and Cordelia Schmid. Groups of adjacent contour segments for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(1):36–51, 2008.

[31] Zhe Lin and Larry S. Davis. Shape-based human detection and segmentation via hierarchical part-template matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(4):604–618, 2010.

[32] Andrew Rabinovich, Andrea Vedaldi, Carolina Galleguillos, Eric Wiewiora, and Serge Belongie. Objects in context. In *ICCV*, pages 1–8, 2007.

[33] Carolina Galleguillos and Serge Belongie. Context based object categorization: A critical survey. Technical report, Computer Science and Engineering, University of California, San Diego, 2008.

[34] Zhuowen Tu. Auto-context and its application to high-level vision tasks. In *CVPR*, 2008.

[35] Santosh Kumar Divvala, Derek Hoiem, James Hays, Alexei A. Efros, and Martial Hebert. An empirical study of context in object detection. In *CVPR*, pages 1271–1278, 2009.

[36] Wei-Shi Zheng, Shaogang Gong, and Tao Xiang. Quantifying contextual information for object detection. In *ICCV*, pages 932–939, 2009.

[37] Bangpeng Yao and Li Fei-Fei. Modeling mutual context of object and human pose in human-object interaction activities. In *CVPR*, pages 17–24, 2010.

[38] Leonid Karlinsky, Michael Dinerstein, Daniel Harari, and Shimon Ullman. The chains model for detecting parts by their context. In *CVPR*, pages 25–32, 2010.

[39] Yong Jae Lee and Kristen Grauman. Object-graphs for context-aware category discovery. In *CVPR*, pages 1–8, 2010.

[40] Carolina Galleguillos, Brian McFee, Serge J. Belongie, and Gert R. G. Lanckriet. Multi-class object localization by combining local contextual interactions. In *CVPR*, pages 113–120, 2010.

[41] Mikel Rodriguez, Ivan Laptev, Josef Sivic, and Jean-Yves Audibert. Density-aware person detection and tracking in crowds. In *ICCV*, pages 2423–2430, 2011.

[42] William Robson Schwartz, Aniruddha Kembhavi, David Harwood, and Larry S. Davis. Human detection using partial least squares analysis. In *ICCV*, pages 24–31, 2009.

[43] Aniruddha Kembhavi, David Harwood, and Larry S. Davis. Vehicle detection using partial least squares. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(6):1250–1265, 2011.

[44] Juergen Gall and Victor Lempitsky. Class-specific hough forests for object detection. In *CVPR*, 2009.

[45] Vladimir N. Vapnik. *The nature of statistical learning theory 2nd edition*. Springer, 2000.

[46] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[47] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large Margin Rank Boundaries for Ordinal Regression*, chapter Advances in Large Margin Classifiers, pages 115–132. MIT Press, Cambridge, MA, 2000.

[48] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[49] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, pages 807–814, 2007.

[50] Vojtech Franc and Sören Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML*, pages 320–327, 2008.

[51] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:xx–xx, 2011.

[52] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, pages 1458–1465, 2005.

[53] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[54] Liefeng Bo and Cristian Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *NIPS*, pages 135–143, 2009.

[55] Mehmet Gonen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.

[56] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[57] Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani. 1-norm support vector machines. In *NIPS*, 2003.

[58] Gleen M. Fung and Olvi L. Mangasarian. A feature selection newton method for support vector machine classification. *Computational Optimization and Application*, 28:185–202, 2004.

[59] Yuh-Jye Lee and Olvi L. Mangasarian. Rsvm: Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

[60] S. Sathiya Keerthi, Olivier Chapelle, and Dennis DeCoste. Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

[61] Mingrui Wu, Bernhard Scholkopf, and Gokhan Bakir. A direct method for building sparse kernel learning algorithms. *Journal of Machine Learning Research*, 7:603–624, 2006.

[62] Thorsten Joachims and Chun-Nam John Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76:179–193, 2009.

[63] Subhransu Maji, Alexander C. Berg, and Jitendra Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.

[64] Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34:480–492, 2012.

[65] Jorome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28:337–407, 2000.

[66] Robert E. Schapier, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[67] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[68] Chunhua Shen and Hanxi Li. On the dual formulation of boosting algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2216–2231, 2010.

[69] Jianxin Wu, S. Charles Brubaker, Matthew D. Mullin, and James M. Rehg. Fast asymmetric learning for cascade face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(3):369–382, 2008.

[70] Chunhua Shen, Peng Wang, and Hanxi Li. Lacboost and fisherboost: Optimally building cascade classifiers. In *ECCV (2)*, pages 608–621, 2010.

[71] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Cost-sensitive boosting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):294–309, 2011.

[72] Rong Xiao, Long Zhu, and HongJiang Zhang. Boosting chain learning for object detection. In *ICCV*, pages 709–715, 2003.

[73] Lubomir D. Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *CVPR (2)*, pages 236–243, 2005.

[74] Rong Xiao, Huaiyi Zhu, He Sun, and Xiaoou Tang. Dynamic cascades for face detection. In *ICCV*, pages 1–8, 2007.

[75] Minh-Tri Pham, V-D. D. Hoang, and Tat-Jen Cham. Detection with multi-exit asymmetric boosting. In *CVPR*, 2008.

[76] Bastian Leibe, Ales Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77(1-3):259–289, 2008.

[77] Alain Lehmann, Bastian Leibe, and Luc J. Van Gool. Fast prism: Branch and bound hough transform for object class detection. *International Journal of Computer Vision*, 94(2):175–197, 2011.

[78] Markus Weber, Max Welling, and Pietro Perona. Unsupervised learning of models for recognition. In *ECCV*, pages 18–32, 2000. ?????

[79] Robert Fergus, Peter Perona, and Andrew Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, volume 2, pages 264–271, 2003.

[80] Pedro F. Felzenszwalb, Ross B. Girshick, David A. McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[81] Ross B. Girshick, Pedro Felzenszwalb, and David Mcallester. Object detection with grammar models. In *Advances in Neural Information Processing Systems 24*, pages 442–450, 2011.

[82] Nima Razavi, Juergen Gall, and Luc J. Van Gool. Backprojection revisited: Scalable multi-view object detection and similarity metrics for detections. In *ECCV (1)*, pages 620–633, 2010.

[83] S. Charles Brubaker, Jianxin Wu, Jie Sun, Matthew D. Mullin, and James M. Rehg. On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3):65–86, 2008.

[84] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *CVPR*, pages 73–80, 2010.

[85] Sudheendra Vijayanarasimhan and Kristen Grauman. Efficient region search for object detection. In *CVPR*, pages 1401–1408, 2011.

[86] Esa Rahtu, Juho Kannala, and Matthew Blaschko. Learning a category independent object detection cascade. In *ICCV*, 2011.

[87] Christoph H. Lampert, Matthew B. Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.

[88] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *NIPS*, pages 561–568, 2002.

[89] Paul A. Viola, John C. Platt, and Cha Zhang. Multiple instance boosting for object detection. In *NIPS*, 2005.

[90] Piotr Dollár, Boris Babenko, Serge J. Belongie, Pietro Perona, and Zhuowen Tu. Multiple component learning for object detection. In *ECCV*, volume 2, pages 211–224, 2008.

[91] Zhe Lin, Guang Hua, and Larry S. Davis. Multiple instance feature for robust part-based object detection. In *CVPR*, 2009.

[92] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. *Semi-supervised learning*. MIT press Cambridge, MA:, 2006.

[93] Meng Wang and Xiaogang Wang. Automatic adaptation of a generic pedestrian detector to a specific traffic scene. In *CVPR*, 2011.

[94] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(5):854–869, 2007.

[95] Nima Razavi, Juergen Gall, and Luc Van Gool. Scalable multi-class object detection. In *CVPR*, 2011.

[96] Patrick Ott and Mark Everingham. Shared parts for deformable part-based models. In *CVPR*, 2011.

[97] Ruslan Salakhutdinov, Antonio Torralba, and Josh Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, pages 1481–1488, 2011.

[98] Yusuf Aytar and Andrew Zisserman. Tabula rasa: Model transfer for object category detection. In *ICCV*, 2011.

[99] Guo-Jun Qi, Charu Aggarwal, Yong Rui, Qi Tian, Shiyu Chang, and Thomas Huang. Towards cross-cateogory knowledge propagation for learning visual concepts. In *CVPR*, 2011.

[100] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, 2009.

[101] Sung Ju Hwang, Fei Sha, and Kristen Grauman. Sharing features between objects and their attributes. In *CVPR*, 2011.

[102] Kristin P. Bennett and Emilio Parrado-Hernandez. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7:1265–1281, 2006.

[103] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[104] T. Joachims. *Making Large-Scale SVM Learning Practical*, chapter Advances in Kernel Methods - Support Vector Learning, pages xx–xx. MIT Press, 1999.

[105] Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9:1369–1398, 2008.

[106] Thorsten Joachims. Training linear svms in linear time. In *ACM SIGKDD*, 2006.

[107] Rong-En Fan, Pai-Hsuen Che, and Chih-Jen Li. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[108] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[109] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77:27–59, 2009.

[110] Matthew B. Blaschko and Christoph H. Lampert. Learning to localize objects with structured output regression. In *ECCV*, volume 1, pages 2–15, 2008.

[111] Chang Huang, Haizhou Ai, Bo Wu, and Shihong Lao. Boosting nested cascade detector for multi-view face detection. In *ICPR (2)*, pages 415–418, 2004.

[112] Peng Wang, Chunhua Shen, Nick Barnes, Hong Zheng, and Zhang Ren. Asymmetric totally-corrective boosting for real-time object detection. In *ACCV*, volume 1, pages 176–188, 2010.

[113] Chris J.C. Burges. Simplified support vector decision rules. In *ICML*, 1996.

[114] Jinbo Bi, Tong Zhang, and Kristin P. Bennett. Column-generation boosting methods for mixture of kernels. In *ACM SIGKDD*, 2004.

[115] Olivier Duchenne, Armand Joulin, and Jean Ponce. A graph-matching kernel for object categorization. In *ICCV*, pages 1792–1799, 2011.

[116] Lubomir D. Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, pages 1365–1372, 2009.

[117] Tae-Kyun Kim and Roberto Cipolla. Mcboost: Multiple classifier boosting for perceptual co-clustering of images and visual features. In *Advances in Neural Information Processing Systems 22*, pages 841–848, 2009.

[118] Devi Parikh and C. Lawrence Zitnick. Find the weakest link in person detectors. In *CVPR*, 2011.

[119] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, 2nd Edition*. Springer-Verlag, 2008.

[120] Zhouyu Fu, Antonio Robles-Kelly, and Jun Zhou. Milis: Multiple instance learning with instance selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):958–977, 2011.

[121] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, 2011.

[122] Xiaoyu Wang, Tony X. Han, and Shuicheng Yan. An hog-lbp human detector with partial occlusion handling. In *ICCV*, pages 32–39, 2009.