COMBINING SIMPLE TRACKERS USING STRUCTURAL SVMS FOR OFFLINE SINGLE OBJECT TRACKING

by

Bahman Yari Saeed Khanloo

B.Sc, University of Tehran, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2010

APPROVAL

Name:	Bahman Yari Saeed Khanloo
Degree:	MASTER OF SCIENCE
Title of thesis:	COMBINING SIMPLE TRACKERS USING STRUCTURAL SVMs FOR OFFLINE SINGLE OBJECT TRACKING
Examining Committee:	Dr. Anoop Sarkar Chair
	Dr. Gregory P. Mori, Senior Supervisor
	Dr. Ghassan Hamarneh, Supervisor
	Dr. Ze-Nian Li, SFU Examiner

Date Approved: Aug 10th 2010 Defence date: Aug 05th 2010

Abstract

We introduce MMTrack, our single-target tracking system, that combines the idea of cluster-based and adaptive appearance modeling. MMTrack uses SVMs for linearly aggregating simple trackers. We focus on modeling tracking as a structured output prediction task where the goal is to find a sequence of interdependent locations of the target given a video. Since trajectories are difficult to characterize and that "bad" trajectories are (usually) not given, we require a principled way for automatically generating them. Following recent advances in machine learning, we discriminatively learn the tracking task by first generating bad trajectories and then employing a max-margin criterion to learn how to distinguish among ground truth trajectories and all other possibilities. Our framework for tracking can be of general interest since one can add or remove trackers easily to obtain a customized tracker with desired properties. Our method enjoys robustness against occlusion, drift and appearance change. We applied our framework to single pedestrian tracking and experimentally demonstrated the effectiveness of our method on a real-world data set by achieving comparable results to the state-of-the-art tracking systems.

""If you are not falling, you are not trying."

— SONNIE TROTTER

Acknowledgments

I would like to take advantage of this opportunity and acknowlede numerous people who were influential during my studies and scholarly work at Simon Fraser University. First of all, I am grateful to my supervisor Greg Mori for the patience, support, deep insights, encouragement, invaluable directions and thoughtful omissions. I would like to thank Kevin Murphy who introduced me to probabilistic machine learning and Bob Hadley whose insights in neural networks inspired me a lot. I have been fortunate to work in the friendly environment of Vision and Media Lab (VML) where constructive and stimulating ideas from faculty and student colleagues were always of help. Specifically, I wish to express my gratitude to Mohammad Norouzi, Mani Ranjbar and Yang Wang for their generous discussions, genuine directions and friendly support. Also, I warmly thank my colleague Ferdinand Stefanus for his honest cooperation and useful discussions. Finally, I appreciate valuable comments by the committee Ze-Nian Li, Greg Mori and Ghassan Hamarneh and extend thanks to Anoop Sarkar for taking the time to serve as chair for my defense seminar.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computers were built to help make repetitive and time-consuming tasks easier to perform for humans. Computer scientists, however, have been interested in developing systems than can accomplish complicated tasks with minimum amount of supervision. In other words, people have been aiming at incorporating intelligence in their systems. In fact, the grand goal of artificial intelligence is to build intelligent machines i.e. machines that can ideally decide and act as an intelligent human being would. Since describing and formulating real-world problems has proven to be unpredictably challenging, we are often concerned with building intelligence by having machines "learn" to solve the tasks of interest by looking at problem-solution instances.

Generally speaking, computer vision is concerned with understanding images and videos. Aside from extracting problem-specific features and representations from images and videos and devising algorithms that can handle the related tasks, computer vision researchers have recently been able to successfully employ learning algorithms in this area. Hence, many problems have often been redefined or revisited from the learning point of view. For example, object recognition problem is sometimes defined as the problem of learning to recognize objects. Although irregularities or rare characteristics can be of interest, the learning perspective of vision originates from the belief that similar events tend to have specific patterns or statistics that implicitly reflect their commonalities. Interestingly, the viewpoint of learning to perform vision tasks has brought up many interdisciplinary challenges and issues that has motivated contributions from both the theoretical learning and practical vision communities.

Here, we are interested in applying computer vision in traffic monitoring and analysis of public transportation. Our work is motivated by the fact that motion statistics and behavioural patterns of people and pedestrian-vehicle interactions on the roads can help improve civil architecture and

road safety. Since gathering such statistics is extremely costly and tedious, our long term goal is to develop reliable systems that can automatically perform these tasks. Hence, we seek to devise algorithms that, given a recorded video of the settings of interest, summarize the activities and motion patterns of people and their interaction with vehicles. Specifically, we focus on tracking the pedestrians that show up in a given video sequence. This would enable us model pedestrian vehicle interactions as well as pedestrian movement patterns from sidewalk to road and vice versa. Following the previously mentioned learning perspective of vision problems, we try to learn to perform tracking by appropriately representing the trajectories and learning to differentiate among good and bad ones given their features. This is inspired by the observation that complex objects, such as trajectories, can be compactly represented using their statistics. For instance, a trajectory can be described by the expectations of patterns of movements and changes in the appearance of the object being tracked. The complexity of objects of interest, on the other hand, suggests using multiple trackers. Therefore, we use a classifier in order to aggregate multiple trackers and learn to characterize the difference between the representations of good and bad trajectories so as to be able to predict a good trajectory by choosing one that respects the expected distributions most. Although characterizing trajectories brings up challenges and issues and learning and inference is costly, we benefit from the added flexibility in terms of features and gain good performance in pedestrian tracking and hope our work would help understand related problems better.

The thesis is organized as follows. The remainder of introduction briefly describes pedestrian detection and tracking and their connection. Next, having motivated tracker combination approaches, we explain our features and learing and inference and summarize the contribution. Chapter 2 discusses the previous work and recent advances in learning functional dependencies in structured spaces that extend them. Chapter 3 briefly reviews some background knowledge that our learning and inference schemes build upon. Chapter 4 provides detailed information about our tracking system. Chapter 5 presents experimental evaluations and Chapter 6 concludes the thesis.

1.1 Pedestrian Detection

Object detection generally refers to the problem of localizing objects of a specific category (such as cars, buildings or pedestrians) in an image or a video. Since building models that can faithfully describe objects is difficult, we are often interested in finding rectangular bounding boxes that enclose the region of the image that belongs to the object of interest such that the center of the object coincides with the center of the box. This is usually done by sliding a window all over the image

and examining whether that window contains an object belonging to the desired category.

Object detectors are often classifiers that decide upon existence of object of interest based on the evidence in a given bounding box. Pedestrian detectors usually perform similarly to detectors designed for other object categories and differ only in the features they employ. Although we do not aim to build a pedestrian detector, we use a state-of-the-art detector [13] in order to initiate our tracking algorithm.

Detection is challenging a problem because objects may be occluded (by objects in the same or different category) or appear in different angles, sizes or scales. They can be of non-rigid or articulated nature (e.g. human body), have a complex shape or pose or even be irregular i.e. have an unusual appearance (for example a person may be missing a leg or wearing a backpack). Moreover, general challanges of computer vision problems such as image noise, differences in scene illumination and information loss in 3D to 2D projection are also to be dealt with.

1.2 Pedestrian Tracking

Object tracking, in its simplest and most general setting, is the task of following an object of interest throughout a video sequence and obtaining its trajectory while it is within the field of view or until some other criterion is satisfied. Tracking is an important computer vision task with numerous applications such as human tracking, motion-based recognition, automated navigation, human-computer interaction, traffic monitoring and video surveillance [44].

Tracking and detection are tightly coupled tasks that can potentially help one another. One can think of tracking as a sequence of detections [27] with some considerations. Namely, motion, time coherence and color are often ignored in object detection. Moreover, since time comes into play in tracking, one has to handle identity preserving issues as many instances of the same category can show up in a video sequence simultaneously for a considerable amount of time.

The tracking problem is very challenging in general due to appearance change, camera motion and the motion of object of interest. Needless to mention that all previously noted issues discussed under object detection also come up in the tracking problem in more or less similar way.

We will be making a few simplifying assumptions to narrow down the problem and to make it tractable. We are interested in the problem of tracking an object in a video recorded using a stationary camera where the initial location of the object is given by some detector which is assumed to be perfectly reliable. Presuming that a mechanism for performing multi-object tracking exists,

we focus our attention on building a good single-object tracking system. Finally, instead of frame-to-frame tracking, we choose to estimate the trajectory of the object throughout the whole video sequence at once hoping that we can gain robustness against occlusion and compensate for simplifications made in our model and respresentations. Although our tracking framework is general, we make it specific for pedestrian tracking by using appearance and motion models that are designed to describe pedestrians.

1.3 Combining Trackers

Building a good tracker that can do well in all the possible complicated settings turns out to be impractical. However, a common approach for overcoming some of the above mentioned challenges is to introduce a number of different trackers that either individually or in groups explain different part of the problem. Most tracking systems amount to defining a set of features that best describe the appearance of the object along with a motion model that explains the patterns of movements between consecutive frames. The appearance and motion models are usually treated separately by tracking algorithms, using independent components whose parameters are set or learned independently (e.g. Haar-like features for appearance model and constant velocity dynamics as the motion model [42], or Eigenbasis appearance model-Brownian motion model pair [31]).

Further, choosing an appearance model is itself challenging due to appearance change of the target and potential similarity in appearance with other objects. Ramanan et al. [30] consider a static approach to appearance modeling using clustering while Collins et al. [9] use online adaptive appearance modeling. A good appearance model should be able to strike a balance between resistance to drift and adaptation of the object's appearance over time. This suggests that a combination of static and continually updated cues may help a tracker achieve superior performance. It has been shown that combining different cues helps improve tracking performance, if it is done in a principled manner [35, 27]. Note that aside from multiple trackers, [27] includes a segmentation i.e. a delineation of boundaries for obtaining better appearance models (which is not always practical due to view limitations or background similarities etc.). Many existing multi-cue tracking algorithms, however, combine the cues with either fixed weighting [5, 34, 33], or update the weights according to a heuristically-selected measure [23, 26].

Another approach that has gained popularity recently is to use some feature selection criteria to select the features that best discriminate the appearance of the object from its surroundings. In [9], a Fisher-like criterion is used to select the most discriminative features at each frame, whereas

in [3], a boosting mechanism is used to select the best features from a fixed pool of features. Both methods, however, use only one type of feature (linear combination of RGB channels for [9], and Haar-like features for [3]). Considering the fact that different feature types help and that the features are not independent from each other necessarily, we try to combine various features and determine their relative contribution in a principled way.

1.4 Features

In general, learning, inference and features are closely related to each other. Often, it is very difficult to consider the wide range of possibilities of these components since exhaustively trying all the combinations is impractical for most of the frameworks and real-world problems. Therefore, we usually choose a learning and inference framework and try to extract features that are suitable for both the problem and the framework or vice versa. Here, we are interested in exploring the former approach and define our task-relevant features accordingly. It is known that the choice of "good" features has an important contribution towards performance. Yet, we design a set of features in such a way that they fit our specific framework. More precisely, we require our features not only to be informative but also to respect the nature of our training procedure. In what follows, we briefly describe the intuitions behind our choice of features and then explain the roles of the features and associate them with their names.

Our goal is to combine the idea of constant appearance modeling [30] and adaptive appearance modeling by striking a balance between trackers representing each of them. Specifically, we introduce a component that forces constant appearance and another that models the appearance change over time. Also, we use trackers built on top of hierarchical color histograms that describe how the histograms of different parts of the bounding box enclosing the pedestrian deviate from their mean over time. Note, however, that we are not interested in recognition (i.e. learning to describe how pedestrians tend to look like) as a pedestrian detector would. Instead, we use information from a reliable detector to help us with localization.

We expect the Histogram of Oriented Gradients (HOG) feature to be of help to the system in discriminating between pedestrians and non-pedestrian objects (e.g. car, trees, etc.). Color histogram distance features are used to provide information about the deviation of appearance of the tracked pedestrian. Appearance template features, on the other hand, give clues about a particular pedestrian at a finer level than the histogram distance features. Further, other than the static histogram distance features, another type of appearance template feature is used to explain expected patterns of

appearance updates for pedestrians. With the combination of both static and continually-updated appearance model, we hope to achieve a good balance between resistance to track drift and adaptation to a pedestrian's varying appearance throughout its trajectory.

HOG Score

We use the output of the linear SVM classifier that operates on HOG [13] as a feature to help our system differentiate between pedestrians and other objects. We trained the SVM classifier to detect pedestrians in top-down view so as to make it suitable for our particular experimental setup. The detection window size is set to 48x112 and the detection is performed on a pyramid built on the input image with its scale varying up to 130% of the original resolution. For each pixel, we take the maximum SVM score over all scales resulting in a score map where the peaks vote for presence of pedestrians. We then normalize these scores so they fall within the range [0,1] and use the final map as our feature. Figure 1.1 illustrates an input image along with the corresponding normalized HOG score map.

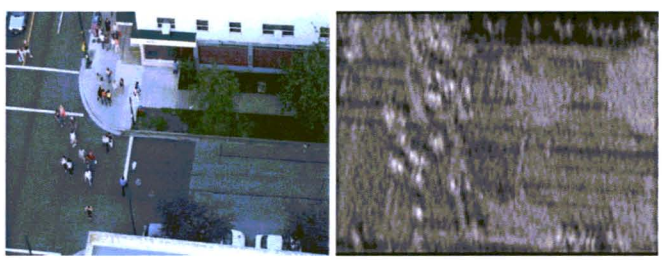

Figure 1.1: A frame and its corresponding HOG feature map.

Color Histogram Distance

Although HOG scores can help the system differentiate between pedestrians and other objects, they are not informative in distinguishing among different pedestrians, as many pedestrians will potentially have high HOG scores. Thus, features that convey identity i.e. features that can uniquely resprsent the appearance of a pedestrian are needed. We incorporate such features using a static color histogram model obtained from clustering. The main idea here is that by clustering the histograms obtained from bounding boxes around the pedestrians throughout the video, we can gain a good insight into the average appearance statistics of each of the people. Thus, we will be able to track

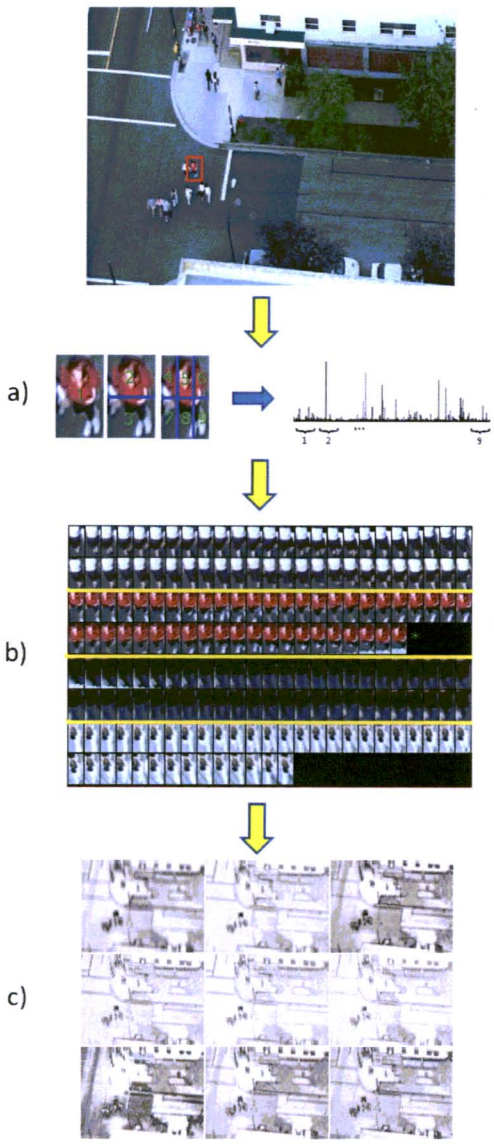

Figure 1.2: Color histogram distance features generation: **a)** Nine histograms are computed over sections of the detected person's bounding box and the resulting histograms are concatenated to give a hierarchical description. **b)** all such histograms are clustered. **c)** Histogram distance maps are then generated for every frame by sliding a window and computing the χ^2-distance between the histogram of each of the sections of the window to the cluster mean to which the target belongs.

the change instead of trying to learn the appearance and so we would be able to obtain a simple yet effective appearance model. Note that static appearance features in our tracker contributes towards gaining resistance againts drift that often occurs in tracking systems with dynamically-updated appearance models.

The generation of the color histogram distance features is as follows. Based on the image evidence inside the bounding box obtained from a HOG detection, nine histograms are computed over different sections of a pedestrian's body as depicted in the second column of Figure 1.2. Each histogram consists of 30 bins with 10 bins for each of the R, G and B channels. These nine histograms are then concatenated together to give one histogram characterizing the person's appearance. Next, we cluster all the instances of histograms of all the people in the video using the mean-shift clustering algorithm. We then represent the target pedestrian using the mean of the cluster to which it belongs. Finally, we compute one histogram distance map for each of the nine body sections by computing, at each pixel location, the χ^2-distance between the histogram built using the image observation within the corresponding section of the bounding box centered at that pixel and the mean of the cluster to which the target belongs. The resulting maps have low values in areas with similar color to the target person's and high values elsewhere. We efficiently compute the histogram distance maps using the integral histogram trick [28].

Appearance Templates

Besides a person's color histogram distance maps, we use appearance templates to describe the person's appearance. We use two templates: a template obtained from the image patch inside the bounding box surrounding the given location of the target in the initial frame, and another template obtained similarly from the previous frame. The initial template implements a constant appearance model which is used to provide a fixed reference to the person's appearance similar to the cluster center of the color histograms. However, the initial appearance template describes the person's appearance at a finer level of detail than the histogram distance features. This template acts as a memory template which ensures that the tracker does not completely forget about the appearance of the target when it first showed up. On the other hand, the previous frame template incorporates the idea of adaptive appearance modeling because it mimics the appearance adaptation mechanism by encoding the expected amount of frame-to-frame change of the tamplate during the inference, which helps the system cope with some degree of object appearance changes over time.

Distance maps are computed for each of the templates by sliding each template over the current frame, and computing the sum of absolute pixel value differences in all three color channels of each

pixel belonging to the template, which is efficiently performed using a modified integral image trick [41]. These distance maps are then normalized to fall in the rage of [0,1].

1.5 Learning and Inference

As noted before, we are interested in exploring learning perspective of the tracking problem. In fact, since we believe that temporal information is important when modeling tracking, we are concerned with the problem of learning to predict trajectories as structured objects. Although we do not build upon probabilistic interpretations, we will be using a simple probabilistic temporal model in order to define our learning and inference procedures.

We propose to use a large margin criterion for learning. In our formulation, having considered each feature as a tracker, we define our tracker to be a linear combination of individual trackers whose relative contributions i.e. the weights are jointly determined. The idea is to induce an appropriate features space where any given trajectory can be summarized as a point and use a Support Vector Machine (SVM) in order to learn to produce good trajectories. Therefore, tracking amounts to performing an inference in our temporal model given the model parameters.

1.6 Contribution

The main contribution of this thesis is to employ structural support vector machines in the context of object tracking. Structural SVM is a recent development in machine learning that generalizes multi-class Support Vector Machine formulation to deal with interdependent and structured variables [37, 39]. Object tracking can be formulated as a structured output prediction problem, as it tries to find the sequence of coordinates that best explain input features. This is verified by the observation that the locations of an object in consecutive frames must be close to each other as an object is assumed to have physically restricted movements. In addition, by formulating the problem as a tree-structured Conditional Random Field (CRF), we will show that Structural SVM also provides an intuitive and principled way to treat the feature representation and motion model in a unified way while jointly learning the relative contributions of multiple cues.

The tree-structured CRF model and the inference scheme that we adopt is similar to the model adopted in [5]. However, our objective is not to build a multi-target tracker but rather to define a principled way to combine different cues. Berclaz et al. [5] combine simple cues such as ground plane occupancy and color model by treating them equally (i.e. using fixed weighting), whereas we try to

find weights that represent relative contributions of features. In a way, the idea of combining different cues in our framework is closely related to the approach presented in [35]. The main difference is that instead of combining the results of multiple 'observers' (i.e. complete tracking systems each having its own appearance and motion model), we fuse different appearance models and a motion model. Moreover, the parameters for fusing the appearance models and motion model are learned jointly in our case whereas in [35] the observers are combined according to error distributions that are learned independently for each observer.

We adopted our framework to devise a single pedestrian tracking algorithm and demonstrated the effectiveness of our method experimentally on a real-world data set achieving comparable resutls to the state-of-the-art. This work was done in collaboration with Ferdinand Stefanus, Mani Ranjbar, Ze-Nian Li and Greg Mori, parts of which appeared as a publication in [22]. My contribution to this work centered around applying structural support vector machines in the context of single target tracking using different sources of information.

Chapter 2

Previous Work

Supervised approaches for learning discrete labels from given independent objects have been around for six decades now. Support vector machine [11], which maximizes the confidence in classification using the so called *margin* criterion has been very successful in immensely many applications. Aside from well studied theoretical foundations and generalization guarantees, support vector machines enjoy efficiency in learning and prediction while being capable of handling very high dimensional spaces through the *kernel trick*. The SVM recipe for modeling a group of variables i.e. multi-class classification of multiple objects, however, is limited to introducing a single joint label for the entire data set with the number of classes being exponential in the size of the set which is intractable. This means that, in practice, SVM is unable to exploit the sparsity in structure for more efficient search in the parameter space.

Although the general problem of modeling complex objects is difficult, the challenge has been effectively addressed for many simple structures using *probabilistic graphical models* e.g. [4] and [24]. Structured prediction i.e. the idea of combining graphical models and SVMs aiming at getting the best of both frameworks was first investigated in the work Hidden Markov Support Vector Machines (HM-SVM) [2] by Altun et al. which was based on an instance of working set optimization for SVMs. This work was inspired mainly by perceptron-like discriminative training work by Collins [8], learning structured output spaces using joint kernels by Hofmann et al. [15] and the n-best algorithm [7]. In the following, another significant contribution towards combining graphical models and the large margin framework, *Max-Margin Markov Networks* was proposed by Taskar et al. [37] which resulted in a novel polynomial-size formulation. Tsochantaridis et al. [39, 40] and Joachims et al. [20], however, viewed the problem as the problem of learning general functional dependencies in structured output spaces and used the cutting plane method to exploit the structure

and sparsity in the dependency network (of course the cutting plane approach is believed to be inspired by the one best pseudo example approach employed in [2]). This line of work was closely related to the learning framework developed for sequence alignment problem (which appeared also as a technical report before publication of [37]) by Joachims in [18]. Interestingly, these pieces of work later inspired the use of cutting plane method in efficient training of linear binary support vector machines [19]. Notably, a generative approach to the structured prediction problem was recently introduced by Lampert and Blaschko [25] which, taking advantage of the one-class SVM formulation, suggests building models for estimating the support of the joint space instead of modeling the full joint distribution.

In the following, we briefly review some related pieces of previous work. For simplicity of presentation, we start with the probabilistic view of multi-class support vector machines and build on that to elaborate on a widely received structural extension of support vector machines and the cutting plane approach used for training them.

2.1 Multi-Class Support Vector Machines

Support vector machines were originally formulated for 2-class classification. An extension to multi-class SVM, however, is not quite straightforward. A common approach is the so called one-versus-the-rest approach where k independent 2-class SVM classifiers are trained to distinguish among data from their class and data from other $k-1$ classes. Here, we are interested in the approach introduced in [12]. In order to understand the idea, we start by drawing analogies between multi-class *logistic regression* and the multi-class SVMs. In logistic regression for the k-class classification problem, we assign a parameter vector to each class conditional and so we have

$$p(y_i = k|x_i; \mathbf{w}_k) = \exp({\mathbf{w}_k}^T x_i). \tag{2.1}$$

The learning is similar to the binary case and the predictor is derived similarly

$$\hat{y_i} = \max_k p(y_i = k|x_i; \mathbf{w}_k). \tag{2.2}$$

In order to optimize for prediction performance, we are interested in the following constraints

$$\forall i, \forall j \neq k, \quad \frac{p(y_i = k|x_i; \mathbf{w})}{p(y_i = j|x_i; \mathbf{w}_j)} \geq c. \tag{2.3}$$

Note that $\mathbf{w}$ is the concatenation of weight vectors for individual classifiers and the choice of constant c is arbitrary. Thus, taking logs and picking an appropriate constant we obtain

$$\forall i, \forall j \neq k, \quad {\mathbf{w}_k}^T x_i - {\mathbf{w}_j}^T x_i \geq 1. \tag{2.4}$$

In order to come up with a unique solution and to avoid overfitting, we consider a positive constant λ and ℓ_2 regularization of the parameters yielding the following objective

$$\min_{\mathbf{w}} \lambda||\mathbf{w}||_2^2 \tag{2.5}$$

$$s.t. \quad {\mathbf{w}_k}^T x_i - {\mathbf{w}_j}^T x_i \geq 1, \quad \forall i, \forall j \neq k. \tag{2.6}$$

We wish to enforce a solution and hence need non-negative slack variables. However, instead of introducing one slack variable for each constraint as in [43], we modify the constraints in Eq. 2.3 as

$$\forall i, \quad \frac{p(y_i = k|x_i; \mathbf{w})}{\max\limits_{\forall j \neq k} p(y_i = j|x_i; \mathbf{w}_j)} \geq c \tag{2.7}$$

which results in the following program

$$\min_{\mathbf{w},\xi} \tfrac{1}{2}||\mathbf{w}||_2^2 + \tfrac{C}{n}\textstyle\sum_i \xi_i \tag{2.8}$$

$$s.t. \quad \forall i, \forall j \neq y_i \quad {\mathbf{w}_{y_i}}^T x_i - {\mathbf{w}_j}^T x_i \geq 1 - \xi_i, \quad \forall i, \xi_i \geq 0. \tag{2.9}$$

Here, C is a positive constant that requires a balance between the contributions of regularizer and the prediction error. We observe that the objective is identical to SVM formulation. It suggests that, according to the probabilistic perspective noted above (which was discussed in many works including [14]), a valid multi-class SVM formalism can be obtained by writing down the usual SVM objective with the constraints forcing a fixed gap between the correct and incorrect labels for each example.

2.2 Structural Support Vector Machines

As pointed out earlier, the structural extensions of support vector machines aim at enabling SVMs to deal with complex objects. Thus, instead of predicting a single label for each instance, we wish to predict a set of labels for a group of interdependent variables jointly. In structural SVM, we inject the properties of a graphical model of interest into SVM classifier using the features coming from a structural extension of the same model.

An interesting instance of this generic problem is the case where predicting a sequence of variables that describe the same category of events is desired. The structural SVM becomes capable of capturing sequential and possibly overlapping dependencies by mimicing, for instance, the structural extension of logistic regressor i.e. Conditional Random Fields model [24]. For brevity, we restrict

ourselves to problems where input varibles are observed and a set of *shared parameters* are responsible for explaining the input-output pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. Similar to the multi-class formulation in Eq. 2.1 and Eq. 2.3, the prediction rule for the i-th sample can be expressed as

$$\hat{\mathbf{y}}_i = \max_{\mathbf{y}_i} p(\mathbf{y}_i|\mathbf{x}_i; \mathbf{w}), \tag{2.10}$$

where the shared parameters $\mathbf{w}$ are to be estimated such that

$$\forall i, \forall \mathbf{y} \neq \mathbf{y}_i, \quad \frac{p(\mathbf{y}_i|\mathbf{x}_i; \mathbf{w})}{p(\mathbf{y}|\mathbf{x}_i; \mathbf{w})} \geq c. \tag{2.11}$$

Again, taking logs we end up with

$$\forall i, \forall \mathbf{y} \neq \mathbf{y}_i, \quad \log p(\mathbf{y}_i|\mathbf{x}_i; \mathbf{w}) - \log p(\mathbf{y}|\mathbf{x}_i; \mathbf{w}) \geq 1. \tag{2.12}$$

As in Eq. 2.7, we come up with the following constraints

$$\forall i, \quad \log p(\mathbf{y}_i|\mathbf{x}_i; \mathbf{w}) - \max_{\mathbf{y}} \log p(\mathbf{y}|\mathbf{x}_i; \mathbf{w}) \geq 1 \tag{2.13}$$

which essentially results in the method discussed in [2]. The maximization in the above gives the runner up to the ground truth (called *pseudo example* by the authors) which we call *negative example* for consistency.

In order to explain complex objects, we introduce a loss function $\Delta(\mathbf{y}_i, \mathbf{y})$ called *structural loss* that describes how a prediction $\mathbf{y}$ differs from the desired labeling $\mathbf{y}_i$. If we incorporate a suitable loss and replace the log probabilities with their equivalent expression using Eq. 3.5, the normalization terms get cancelled and including the regularizer and slack variables we obtain

$$\min_{\mathbf{w}, \xi} \tfrac{1}{2}||\mathbf{w}||_2^2 + \tfrac{C}{n} \textstyle\sum_i \xi_i, \tag{2.14}$$

$$s.t. \quad \forall i, \xi_i \geq 0, \quad F(\mathbf{x}_i, \mathbf{y}_i; \mathbf{w}) - \max_{\mathbf{y}} F(\mathbf{x}_i, \mathbf{y}; \mathbf{w}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i. \tag{2.15}$$

The above formulation is in fact the so called *structural support vector machines*. Here, the loss function which reflects the quality of classifier in the specific problem at hand, accounts for the fact that the gap i.e. the *margin* between a prediction and its corresponding ground truth must be determined according to the loss incurred when predicting it.

It is worthwhile considering again the constraints in Eq. 2.15. In fact, each of these constraints is implicitly representing exponentially many constraints given by

$$\forall i, \forall \mathbf{y} \neq \mathbf{y}_i, \quad F(\mathbf{x}_i, \mathbf{y}_i; \mathbf{w}) - \max_{\mathbf{y}} F(\mathbf{x}_i, \mathbf{y}; \mathbf{w}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i. \tag{2.16}$$

If we plug them in and use the feature representation Φ, we obtain the formulation in [39] given by

$$\min_{\mathbf{w},\xi} \frac{1}{2}||\mathbf{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \quad s.t. \quad \forall i, \xi_i \geq 0, \tag{2.17}$$

$$\forall i, \forall \mathbf{y} \neq \mathbf{y}_i \quad \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i. \tag{2.18}$$

This method, which was proposed in [37] and [39] known as the *margin rescaling* formulation in the latter, is extensively used since it is simple and the maximization needed to produce the constraints in Eq. 2.15 can be made efficient for many problems. A drawback of this approach is that it may put too much effort on pushing easy negative examples away from the ground truth namely manipulating the margin size for examples that are far from being confusing. For example, a bad example with a large slack might be unnecessarily prioritized resulting in other examples being dominated.

On the contrary, another approach is to maintain a fixed margin as in the general SVM formulation and rescale the slack proportional to the loss. This is known as *slack rescaling* [39] which focuses on difficult negative examples and results in the following optimization problem

$$\min_{\mathbf{w},\xi} \tfrac{1}{2}||\mathbf{w}||_2^2 + \tfrac{C}{n}\textstyle\sum_i \xi_i, \tag{2.19}$$

$$s.t. \quad \forall i, \xi_i \geq 0, \quad \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y}} \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}) \geq 1 - \tfrac{\xi_i}{\Delta(\mathbf{y}_i, \mathbf{y})}. \tag{2.20}$$

To see the differences between these two methods for modeling the *hinge loss*, we observe that in the margin scaling the position of the margin is adjusted whereas in slack scaling the slope is modified and the marigin is fixed:

$$\xi_i{}^s = \max_{\mathbf{y}} \Big(\Delta(\mathbf{y}_i, \mathbf{y})\big(1 - \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y})\big)\Big), \tag{2.21}$$

$$\xi_i{}^m = \max_{\mathbf{y}} \Big(\Delta(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y})\Big). \tag{2.22}$$

The slack ξ^s in slack rescaling formulation is invariant to the loss scale changes. For instance, if we set $\Delta' = \lambda\Delta$ with $\lambda > 0$, then we can get the same solution i.e. the same set of weights by building a new program using Δ' and $C' = C/\lambda$. This is not the case in margin rescaling and one also needs to rescale the feature vectors accordingly. In slack rescaling, the slack is just a function of high scoring examples that reside in the margin area as the components of the right-hand side of Eq. 2.21 are non-negative. Clearly, in margin rescaling this does not hold and an example can have a huge loss with features that do not resemble the features of ground truth at all. In practice, however, it turns out that the difference between these methods is not significant experimentally. On the other hand, slack rescaling can be very complicated and expensive and so, one may wish to stick to the margin scaling for many applications.

Although we limited our discussion to posterior probabilities, one can view structural SVM as a powerful framework for learning arbitrary general dependencies that are expressible using tractable probabilistic graphical models.

2.3 Cutting Plane Training of SSVMs

As pointed out earlier, the optimization problems we aim to solve are problematic in that we can not hand them in directly to solvers that employ common methods as the huge number of constraints renders them useless. Therefore, instead of solving the problem with full constraints, we seek to find a polynomially sized subset of constraints such that if we solve for them we can guarantee a predetermined desired accuracy [40].

It is worth considering again the problem setting of interest. One can think of our problem as a weighted instance of one class classification problem [32] where, given a number of positive samples we are interested in localizing (i.e. finding the center of) the positive class by exploring the "nearby" space. Here, we consider a polyhedron $\mathcal{P}$ constructed by querying the cutting plane oracle (explained in Section 3.3) which is updated using potentially more accurate/informative queries as we proceed. The idea is to approximate the borders by having data points vote for the location of the positive set. In order to do so, we examine how much we can move each positive sample according

Algorithm 1 Cutting Plane Training of SSVM

Require: input pairs $(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)$, ε, C
  $S_i = \emptyset, \xi_i = 0, i = 1, ..., n$
  $\mathbf{w} = \mathbf{0}, \mathcal{P} = \emptyset$
  repeat
    for $i = 1, ..., n$ **do**
      $H(\mathbf{y}) = \left\{ \begin{array}{ll} \Delta(\mathbf{y}_i, \mathbf{y}) - \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}) & \text{Margin Scaling} \\ \Delta(\mathbf{y}_i, \mathbf{y})(1 - \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y}_i) + \mathbf{w}^T\Phi(\mathbf{x}_i, \mathbf{y})) & \text{Slack Scaling} \end{array} \right\}$
      $\hat{\mathbf{y}} = \underset{\mathbf{y}}{\text{argmax}}\ H(\mathbf{y})$
      $\xi_i = \max\ (0, \max_{\mathbf{y} \in S_i} H(\mathbf{y}))$
      if $H(\hat{\mathbf{y}}) > \xi_i + \varepsilon$ **then**
        $S_i = S_i \cup \{\hat{\mathbf{y}}\}$
        $\mathcal{P} = \text{update}(\mathcal{P}, \Phi(\mathbf{x}_i, \hat{\mathbf{y}}))$
        $\mathbf{w} \leftarrow \text{aggregate}(\mathcal{P}, \xi)$
      end if
    end for
  until no S_i has changed during iteration

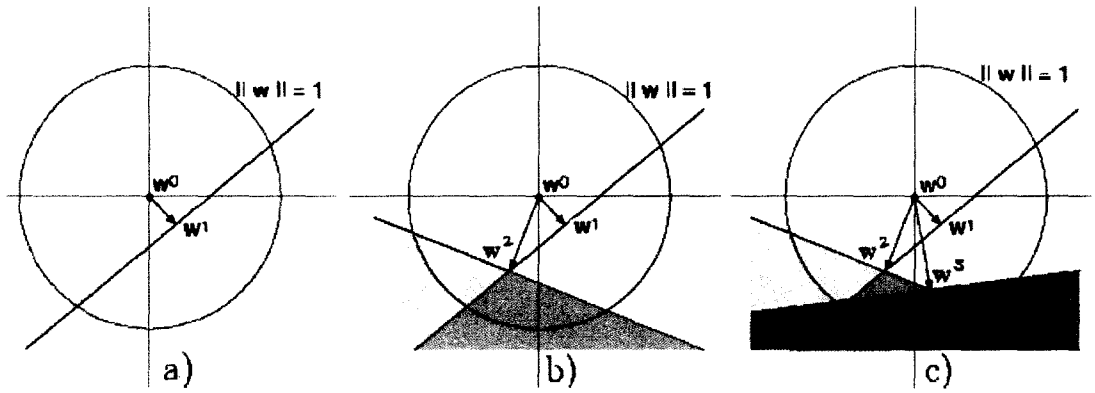

Figure 2.1: A few stages of cutting plane optimization. Shading denotes the current solution and we start with no constraints and $\mathbf{w}_0 = 0$. **a)** First constraint added which cuts off the solution. **b)** Another constraint added and the solution recomputed. **c)** The procedure is repeated until no violation more than ε is detected. Figure taken from [1].

to the displacement cost associated by function $H(.)$. This is done by building vectors $\Delta\Phi_i = \Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \mathbf{y})$ that characterize the displacement for each positive sample, and weighting them with their cost while trying to encourage the informativeness (i.e. prioritize displacements that give negative examples which explore new places and the ones that are explaining finer details). This is implemented using the *aggregate* function as explained in our simplified version of the method in algorithm 1. Roughly speaking, this function expands the positive class boundaries by weighting displacement vectors for each sample and considering the center of the region induced by all such vectors as an estimate for the parameters $\mathbf{w}$.

Figure 2.1 illustrates some consecutive weight updates for the slack scaling variation of the cutting plane optimization. We summarize the procedure as follows. We iterate over the training samples and solve for the examples whose constraint (the cost for the sample introduced by the oracle) is violated by more than the current threshold. We shrink the threshold and obtain a new relaxation using all the constraints added so far and repeat until the desired accuracy ε is reached.

Chapter 3

Preliminaries

3.1 Max-Margin Learning for Structured Spaces

Structured prediction problems are problems where predicting a number of interdependent variables is desirable. The structure can either be modeled in inputs or outputs or both inputs and outputs. Assuming that input and output variable vectors are of length L, we are interested in structured prediction problems where we wish to learn a mapping from an input $\mathbf{x} \in \mathcal{X}^L$ to a discrete output label $\mathbf{y} \in \mathcal{Y}^L$ given training pairs $(\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_n}, \mathbf{y_n})$ drawn from some unknown yet fixed distribution P. Note that a sequence of inputs is to be mapped to a sequence of structured response labels. So, we are dealing with a generalization of multi-class classification. Since solving for one label per structure is impractical and in fact unnecessary for many of the problems, having exploited the structure and the relationships between output labels, we reduce the problem to finding a discriminant function. Such a function is specified as a mapping $F : \mathcal{X}^L \times \mathcal{Y}^L \rightarrow \mathcal{R}$ which basically assigns a real-valued *score* to input-output pairs. Hence, the discriminant function becomes a $\mathbf{w}$-parameterized scoring function expressed by

$$f(\mathbf{x}; \mathbf{w}) = \max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}, \mathbf{y}; \mathbf{w})$$

whose maximum is ideally at the desired output $\mathbf{y}$ for a given input $\mathbf{x}$. Further, one needs to extend zero-one classification loss and introduce the notion of loss for structured prediction. The basic underlying idea is to reflect the fact that different structure choices should be treated according to their correctness using a bounded loss function $\Delta : \mathcal{Y}^L \times \mathcal{Y}^L \rightarrow \mathcal{R}$. This function is expected to convey reliable information about how a prediction $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w})$ differs from the desired labeling $\mathbf{y}$. Then, the goal of learning is to find a function $f(.; \mathbf{w})$ from a certain family of functions that

minimizes the following weighted sum

$$R_{\Delta}(f) = \sum_{\mathcal{X}\times\mathcal{Y}\in S} \Delta(\mathbf{y}, f_{\mathbf{w}}(\mathbf{x}))P(\mathbf{x},\mathbf{y})$$

where $R_{\Delta}(f)$ is the empirical risk on the training sample pairs $S = \{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}\}$ drawn from P. Considering the common i.i.d. assumption, the risk reduces to

$$R_{\Delta}(f) = \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{y}_i, f(\mathbf{x}_i))$$

which is the quantity of interest in SVM learning framework. In fact, the max-margin recipe for learning structured spaces amounts to minimizing an upper bound on the structural loss of the predictor f while avoiding overfitting using the so-called margin criterion.

3.2 Modeling Structure in the Data

In general, the term *model* refers to probabilistic models and *modeling* is the procedure of specifying the relationship and dependencies between the variables according to the nature of the processes to which they are assigned. Here, we are interested in models that use conditional independence for describing the relationships between the variables. Specifically, we define a *Markov network* over the variables so we can easily express the dependencies in terms of a product of factors defined over subsets of these variables.

Consider prediction problems where, instead of a single response variable $y \in \mathcal{Y}$, we are aiming at predicting a set of variables $\mathbf{y} = (\mathbf{y}^{(1)}, ..., \mathbf{y}^{(n)}) \subseteq \mathcal{Y}_1 \times ... \times \mathcal{Y}_n$. For instance, $\mathbf{y}$ can model the trajectory of an object in the context of tracking and each element $\mathbf{y}^{(t)}$ in this variable vector would represent the pixel location of the object at time $t \in \{1, ..., n\}$. The joint space induced by these variables is known as a *structured space* because it is characterized by their interdependence and constraints. Interestingly, by exploiting the dependencies and interactions between the variables, we would not need to model the whole space in practice. For example, in the tracking problem we do not really need to consider the huge space of all possibilities and it is enough, for each pair of consecutive frames, to look at nearby pixels of the hypothesized location because an object can not "fly away".

Structured prediction is the problem of learning to predict structured spaces. Again, we are assuming that input and output variable vectors are of length L. In structured prediction, we are interested in the general problem of learning a mapping from inputs $\mathbf{x} \in \mathcal{X}^L$ to discrete output

labels $\mathbf{y} \in \mathcal{Y}^L$ given a training set $S = \{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}\}$ containing n samples drawn uniformly from some unknown but fixed distribution P. Hence, assuming that P comes from an *exponential family*, we can express it as

$$p(S|\mathbf{w}) = \exp\left[\sum_{i=1}^{n}\Big(\langle \mathbf{w}, \Phi(\mathbf{x}_i, \mathbf{y}_i)\rangle - g(\mathbf{x}_i; \mathbf{w})\Big)\right] \tag{3.1}$$

where $g(.)$ is the *partition function* and $\Phi(\mathbf{x}, \mathbf{y})$ is a *combined feature representation* that encodes the structure in input-output pairs. More precisely, the representation of the joint space respects the independence assumptions of the network of variables on which it is defined. Considering the Markov properties of the network and linearity of the model, each $\Phi(\mathbf{x}_i, \mathbf{y}_i)$ factorizes as

$$\Phi(\mathbf{x}_i, \mathbf{y}_i) = \sum_{c \in C} \Phi_c(\mathbf{x}_i, \mathbf{y}_i). \tag{3.2}$$

where C is the set of maximal cliques defined over the variables and $\Phi_c(.)$ is the vector valued *potential function* which provides a concatenated representation of the contributions of the nodes and edges for clique c in the underlying dependency graph. Interestingly, $\Phi(\mathbf{x}, \mathbf{y})$ is the sample average of the sufficient statistics $\Phi(\mathbf{X}, \mathbf{Y}) = \sum_i \Phi(\mathbf{x}_i, \mathbf{y}_i)$ and we expect $\mathbf{w}$ to reflect this fact appropriately. For example, the maximum-likelihood principle for the likelihood amounts to the following condition

$$E[\Phi(\mathbf{x}, \mathbf{y})] = \Phi(S) = E_S[\Phi(\mathbf{x}, \mathbf{y})] \tag{3.3}$$

whereas the same principle for the likelihood ratios results in conditions on expectations on the difference between sufficient statistics of the distributions that represent individual likelihoods. In other words, in a likelihood ratio setting, $\mathbf{w}$ is supposed to tell us how the average of the values of the features corresponding to each of the distributions differ from one another.

We restrict our attention to conditional models as modeling the input space is burdensome and in fact unnecessary in most of the problems. So, the quantity of interest is

$$p(\mathbf{y}|\mathbf{x}; \mathbf{w}) = \frac{1}{Z(\mathbf{x}; \mathbf{w})} \exp(F(\mathbf{x}, \mathbf{y}; \mathbf{w})) \tag{3.4}$$

where we have defined $F(\mathbf{x}, \mathbf{y}; \mathbf{w}) = \langle \mathbf{w}, \Phi(\mathbf{x}, \mathbf{y})\rangle$ to be a linear *scoring function* that summarizes the joint representation of a given pair $(\mathbf{x}, \mathbf{y})$. Since the model is log-linear, we can write down the log-conditional in the following simple form

$$\log p(\mathbf{y}|\mathbf{x}; \mathbf{w}) = F(\mathbf{x}, \mathbf{y}; \mathbf{w}) - \log Z(\mathbf{x}; \mathbf{w}). \tag{3.5}$$

Note that the normalizing constant Z is only a function of the input which is often cancelled out in cases where likelihood ratio is being modeled.

3.3 The Cutting Plane Method

The *cutting plane method* also known as the *Kelley method* [21] refers to general iterative methods for solving optimization problems with linear inequalities. These methods are appealing since they neither require differentiability of the objective and constraints nor do they need the evaluation of these quantities at each iteration and so they are suitable for problems with large number of constraints. For simplicity, we focus on standard convex minimization problems where the optimal set i.e. the target is given.

In this approach, the objective is to find a convex region $\mathcal{T} \in \mathcal{R}^d$ called the *target* whose description is provided only through a so-called *oracle*. The role of oracle is to determine, given a query point $x \in \mathcal{R}^d$, if it lives in the target or to return a hyperplane called *cut* located between x and $\mathcal{T}$ that helps us narrow down the search by ruling out a halfspace. The hyperplane is parameterized with a and b that must satisfy the following

$$\forall t \in \mathcal{T} \quad a^T t \leq b, \quad a^T x \geq b, \quad ||a||_2 = 1. \tag{3.6}$$

We start by an initial set that is known to contain $\mathcal{T}$, namely we have

$$\mathcal{T} \subseteq \mathcal{P}_0 = \{z | \, A_0 z \leq b_0\}. \tag{3.7}$$

Now we keep sampling from the space bounded in $\mathcal{P}_0$ by querying from the oracle until we come up with k points $x^{(1)}, ..., x^{(k)}$ that do not lie in $\mathcal{T}$. Hence, we obtain the following cutting planes

$$a_i{}^T z \leq b_i, \quad i = 1, ..., k. \tag{3.8}$$

The polyhedron built using these k hyperplanes, denoted by $\mathcal{P}_k$, encapsulates $\mathcal{T}$ namely

$$\mathcal{T} \subseteq \mathcal{P}_k = \{z | \, A_0 z \leq b_0, \quad a_i{}^T z \leq b_i, \quad i = 1, ..., k\}. \tag{3.9}$$

The idea is to obtain a good approximation of the target by tightening the hypothesis $\mathcal{P}$ using nested solutions $\mathcal{P}_0 \supseteq ... \supseteq \mathcal{P}_k \supseteq \mathcal{T}$. If we realize that $\mathcal{P}_k$ is empty we conclude $\mathcal{T}$ is empty and stop. Otherwise, we sample another point inside $\mathcal{P}_k$ and stop if it lies in $\mathcal{T}$ or update the solution using the new point to get $\mathcal{P}_{k+1}$. A simple implementation of the idea [6] is summarized in Algorithm 2.

The informativeness of the cutting planes that we introduce can be captured by the reduction in size of the feasible space. Therefore, an ideal procedure is one that consistently cuts as much as possible from the uncertain region of the search. Since the discarded part of the space is unknown

Algorithm 2 The Generic Cutting Plane Algorithm

Require: An initial polyhedron $\mathcal{P}_0$ containing $\mathcal{T}$
  $k \leftarrow 0$
  loop
    Choose a point $x^{(k+1)}$ in $\mathcal{P}_k$
    Query the oracle at $x^{(k+1)}$
    if $x^{(k+1)} \in \mathcal{T}$ **then**
      Exit Loop
    else
      $\mathcal{P}_{k+1} = \mathcal{P}_k \cap \{z |\ a_{k+1}^T z \leq b_{k+1}\}$
    end if
    if $\mathcal{P}_{k+1} = \emptyset$ **then**
      Exit Loop
    end if
    $k \leftarrow k + 1$
  end loop

a priori, a good query point turns out to be the one that lies in the center of the current polyhedron. Progress measure is often chosen to be the ratio of the volume of two consecutive solutions.

An important concern in this method is that the number of cutting planes grows as we proceed. So, we require good strategies for maintaining a reasonably sized working set. Although redundant constraints (i.e. hyperplanes whose consideration neither changes the hypothesis $\mathcal{P}$ nor the convergance speed) can be exactly identified, most practical implementations employ heuristics such as relevance or ranking to prune the working set.

Another important aspect of the algorithm is the stopping condition. A suitable approach, which is usually used in convex minimization, is to keep record of the value and the location of the optimal point for the objective and use its difference to the best known lower bound obtained in the previous step as the stopping criterion.

Chapter 4

Structural Support Vector Machines for Tracking

In this section, we introduce our tracking system called MMTrack in the context of pedestrian tracking. The system is comprised of three main components: constant appearance model, adaptive appearance model and motion model. Constant appearance model is used to represent the appearance of the target pedestrian whereas adaptive model is used to distinguish among the tracked object and other objects. Also, the motion model favors specific movement patterns from one frame to another. A large margin learning approach combines these three cues by learning weights associated to each of the components. Finally, the learned model is used to estimate pedestrian trajectories.

The rest of this section is organized as follows. Section 4.1 describes our trajectory modeling approach. We explain our max-margin formulation for tracking in Section 4.2 and outline our inference schemes in Section 4.3.

4.1 Trajectory Modeling

As pointed out earlier, we are interested in *offline tracking* where the goal is to obtain the whole trajectory in the entire sequence given an initial location. This is in contrast to *online* tracking algorithms that greedily pick the next best location of the object at each frame. Thus, the tracking problem in this setting is formulated as one of finding the optimal trajectory $\mathbf{y} = (\mathbf{y}^{(1)}, ..., \mathbf{y}^{(T)})$ with the starting location $\mathbf{y}^{(1)}$ and the image sequence $\mathbf{x} = (\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)})$ given. We use a simple tree-structured model as illustrated in Figure 4.1 and define a trajectory to be optimal if it scores the

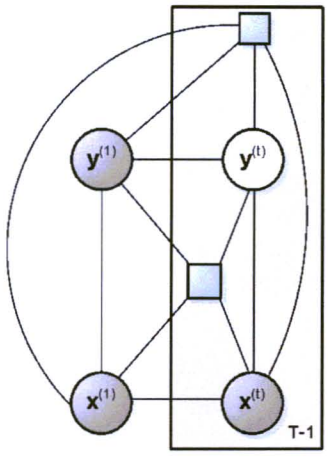

Figure 4.1: Our model: a tree-structured CRF with the initial label and all the inputs observed. The rectangles show the factors of the graph and the plate notation refers to $T-1$ replications of the structure for each frame that give the temporal model.

highest among all possible tracks according to this model.

Our scoring function is a mapping in the form of $F(\mathbf{x},\mathbf{y};\mathbf{w}) : \mathcal{X}^T \times \mathcal{Y}^T \rightarrow \mathcal{R}$ that maps a sequence of frames $\mathbf{x} = \{\mathbf{x^{(1)}},...,\mathbf{x^{(T)}}\}$ and a trajectory $\mathbf{y} = \{\mathbf{y^{(1)}},...,\mathbf{y^{(T)}}\}$ to a real number. Each location $\mathbf{y^{(t)}}$ is a discrete variable which is to be assigned to one of the image pixels and $\mathbf{w}$ is a set of weights that parameterize the features extracted from the frames. The scoring function of this model is decomposed into two contributions: transition model and observation model. The transition model in our problem is summarized by the *motion model* which describes the spatial relationship between the locations of the target in two consecutive frames. The observation model is a measure of compatibility between a location and the observed features at that pixel location. We define the score of a trajectory as

$$F(\mathbf{x},\mathbf{y};\mathbf{w}) = \sum_{t=2}^{T} F_{\mathcal{T}}(\mathbf{y}^{(t-1)},\mathbf{y}^{(t)};\mathbf{w}_{\mathcal{T}})$$
$$+\sum_{t=2}^{T} F_{\mathcal{O}}(\mathbf{x}^{(1)},\mathbf{x}^{(t-1)},\mathbf{x}^{(t)},\mathbf{y}^{(1)},\mathbf{y}^{(t-1)},\mathbf{y}^{(t)};\mathbf{w}_{\mathcal{O}}) \tag{4.1}$$

where $F_{\mathcal{T}}(\cdot)$ and $F_{\mathcal{O}}(\cdot)$ are linear models describing transition and observation contributions respectively. These potential functions are parameterized by $\mathbf{w}_{\mathcal{T}}$ and $\mathbf{w}_{\mathcal{O}}$ whose concatenation we denote by $\mathbf{w}$.

Observation Model

The observation model includes several features whose weighted combination votes for the presence of the target pedestrian. These features include HOG score that helps with discriminating among humans and other objects and, color histogram distance and appearance templates that describe how the pedestrian looks like and how its look varies over time. Thus, the observation model at time t decomposes into the following contributions

$$\begin{aligned} F_{\mathcal{O}}(\cdot;\mathbf{w}_{\mathcal{O}}) &= {\mathbf{w}_{\mathcal{H}}}^T \Phi_{\mathcal{H}}(\mathbf{x}^{(t)},\mathbf{y}^{(t)}) \\ &+ {\mathbf{w}_{\mathcal{C}}}^T \Phi_{\mathcal{C}}(\mathbf{x}^{(t)},\mathbf{y}^{(t)}) \\ &+ {\mathbf{w}_{\mathcal{P}}}^T \Phi_{\mathcal{P}}(\mathbf{x}^{(t-1)},\mathbf{x}^{(t)},\mathbf{y}^{(t-1)},\mathbf{y}^{(t)}) \\ &+ {\mathbf{w}_{\mathcal{F}}}^T \Phi_{\mathcal{F}}(\mathbf{x}^{(1)},\mathbf{x}^{(t)},\mathbf{y}^{(1)},\mathbf{y}^{(t)}) \end{aligned} \tag{4.2}$$

where $\Phi_{\mathcal{H}}(\cdot)$, $\Phi_{\mathcal{C}}(\cdot)$, $\Phi_{\mathcal{P}}(\cdot)$ and $\Phi_{\mathcal{F}}(\cdot)$ denote the potential functions representing HOG score, color distance histogram, and the difference between appearance templates of the previous frame and the first frame to the current frame respectively. We concatenate all the observation weights to give $\mathbf{w}_{\mathcal{O}} = [\mathbf{w}_{\mathcal{C}};\mathbf{w}_{\mathcal{H}};\mathbf{w}_{\mathcal{F}};\mathbf{w}_{\mathcal{P}}]^T$. Intuitively, $\mathbf{w}_{\mathcal{O}}$ weighs the observation features i.e. the trackers to give a map that ideally peaks at the body center of the target pedestrian.

Transition Model

Similar to the observation model, we define the transition model as

$$F_{\mathcal{T}}(\mathbf{y}^{(t-1)},\mathbf{y}^{(t)};\mathbf{w}_{\mathcal{T}}) = {\mathbf{w}_{\mathcal{T}}}^T \Phi_{\mathcal{T}}(\mathbf{y}^{(t-1)},\mathbf{y}^{(t)}), \tag{4.3}$$

where $F_{\mathcal{T}}(\cdot)$ is in fact a symmetric motion model. The motion model discretizes the distance travelled between two consecutive frames into a number of bins that represent concentric circles centered at the previous location. So, we have

$$\Phi_{\mathcal{T}}(\mathbf{y}^{(t-1)},\mathbf{y}^{(t)}) = \text{bin}(d(\mathbf{y}^{(t-1)},\mathbf{y}^{(t)})), \tag{4.4}$$

$$\text{bin}_k(d') = \mathbb{1}_{[\lfloor d' \rfloor = k]}, \quad k = 0,...,\lfloor d_{\max} \rfloor. \tag{4.5}$$

Here, $d(y,y')$ is the Euclidean distance between the $2d$ image locations of y and y', $\mathbb{1}_{[.]}$ is the indicator function and bin$(\cdot)$ acts as a selection operator that generates a vector of length $d_{max}+1$ with all the elements set to 0 except one being 1. The upper bound d_{max} on the travelled distance from one frame to the next one is estimated using the dataset. Note that the symmetric motion model results in $\mathbf{w}_{\mathcal{T}}$ being a disk-like motion prior which is learned jointly with the observation model parameters.

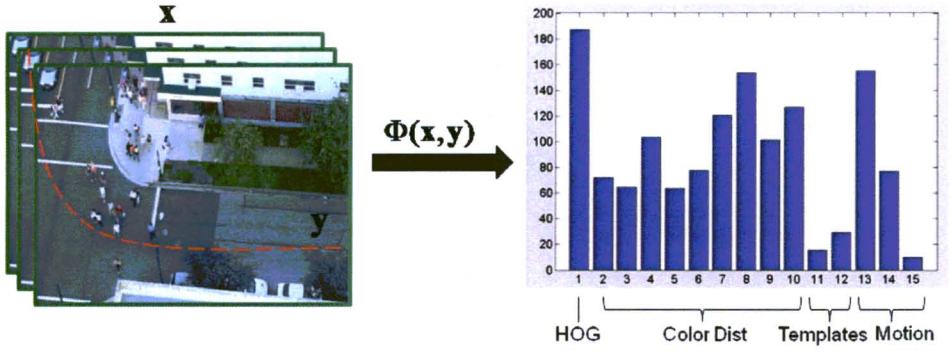

Figure 4.2: A given image sequence $\mathbf{x}$ and trajectory $\mathbf{y}$ and their unnormalized joint representation.

Trajectory Representation

As noted earlier, we require a combined feature representation in order to build our scoring function. We encode a trajectory-video pair $(\mathbf{x}, \mathbf{y})$ using a function $\Phi(.)$, whose components we implicitly introduced previously, that compactly represents their statistics. Figure 4.2 illustrates a ground truth input-output pair and its histogram representaion. Recall that this feature representation is decomposed in the same way that the model parameter vector does, namely $\Phi = [\Phi_{\mathcal{H}}; \Phi_{\mathcal{C}}; \Phi_{\mathcal{P}}; \Phi_{\mathcal{F}}; \Phi_{\mathcal{T}}]^T$. Note that HOG, color distance features and template features are real-valued whereas motion features are of counting nature and hence of type integer.

In this example, we observe that the overall change in color distribution of upper body and top-left part of the body tends to be less compared to other color histogram hierarchies. Also, the amount of change in appearance templates from the first frame to the current one is higher than the change from the previous template to the current one. Moreover, the distribution of the motion counts are as expected considering the following three facts: top-down view of the camera, high frame rate and frequent stops (for short-time groupings, waiting for light, etc).

4.2 Max-Margin Formulation for Tracking

As described earlier, trajectories are difficult to predict as the complexity of their space is exponential in their length. Therefore, the idea is to design a linear function that measures the quality of trajectories and to estimate its parameters such that we can identify good tracks from bad ones by looking at the scores they would end up having according to the scoring function.

Our scoring function is parameterized by a set of weights $\mathbf{w}$ which puts together a variety of desired features linearly. The learning task is to find a set of parameters that best explain the dependencies between image features and trajectories. We use a discriminative approach, namely we try to discriminate between a compatible video-trajectory pair and all other trajectories. Hence, we find a predictor that estimates the best trajectory given an input video by learning a set of parameters that maximize the score of training set examples while pushing down the score of potential runner-ups.

Learning the model parameters in this problem setting is challenging since we do not have negative examples. In other words, we do not know how a "bad" trajectory looks like and more importantly, how it differs from a "good" one because this information is not included in the dataset. Notice that the scoring function in equation 4.1 can be viewed as a $\mathbf{w}$-parameterized discriminant function. Further, the locations in a trajectory are highly interdependent and so we are dealing with a structured output problem. Hence, it is natural to adopt structural support vector machines to jointly estimate the parameters. Here, we consider the notation and viewpoint suggested in [39].

According to the large margin criterion in structural SVM, we require a set of parameters that maximize the score of n given ground truth tracks while pushing away the score of all other trajectories from these maxima and hence the following program

$$\min_{\mathbf{w},\xi} \frac{1}{2}||\mathbf{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \quad s.t. \quad \forall i, \xi_i \geq 0, \tag{4.6}$$

$$\mathbf{w}^T\bar{\Phi}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{w}^T\bar{\Phi}(\mathbf{x}_i, \mathbf{y}) \geq \bar{\Delta}(\mathbf{y}_i, \mathbf{y}) - \xi_i, \tag{4.7}$$

$$\forall i = 1, ..., n, \forall \mathbf{y} \in \mathcal{Y}^T \backslash \mathbf{y}_i.$$

The constant $C > 0$ specifies the relative importance of margin maximization and error minimization which is determined by cross validation. Note that, the constraints can be expressed in various ways among which we are considering the *margin rescaling* formulation. Here, 4.6 guarantees generalization while learning whereas the constraints in 4.7 require the score of ground truth $\mathbf{y}_i$ to be at least as far away from the score of a possibly incorrect trajectory $\mathbf{y}$ as the loss $\Delta(\mathbf{y}_i, \mathbf{y})$ incurred when predicting $\mathbf{y}$. The averaging is performed to make examples with different lengths comparable since in an unnormalized representation, an example may seem unreasonably difficult (or easy) as its location with respect to the hyperplane(s) and hence the shape and location of the feasible region would depend upon the length of the sequence.

The loss function measures the total squared Euclidean distance between corresponding locations in two trajectories:

$$\Delta(\mathbf{y}_i, \mathbf{y}) = \sum_{t \in \mathcal{T}} d^2(\mathbf{y}_i{}^{(t)}, \mathbf{y}^{(t)}). \tag{4.8}$$

In tracking, a target is often considered to be "lost" if the tracker is off by more than a predefined number of pixels ρ. So, one can consider all such trajectories as being equally invalid since they are false tracks anyway. Therefore, we also define a *bounded loss* function which is again sum of individual losses incurred at each frame. The bounded loss at time t is expressed as

$$\Delta_B{}^{(t)}(\mathbf{y}_i, \mathbf{y}) = \begin{cases} \Delta^{(t)}(\mathbf{y}_i, \mathbf{y}) & \text{if } \Delta^{(t)}(\mathbf{y}_i, \mathbf{y}) \leq \rho^2, \\ \rho^2 & \text{otherwise.} \end{cases} \tag{4.9}$$

We use the SVM^{struct} framework[39] to solve this problem. In this approach, we find a subset of inequality constraints in equation 4.7, the most violated ones, and then solve for them such that all the constraints would be violated by no more than a desired precision.

As discussed earlier, since this problem has a huge number of contraints, we resort to the cutting plane method for training. In this approach, for each training pair $(\mathbf{x}_i, \mathbf{y}_i)$ we identify the most confusing constraint i.e. a trajectory that is not only the most appealing one given the current model parameters but also maximizes the loss function. Hence, we seek to find

$$\hat{\mathbf{y}} = \underset{\hat{\mathbf{y}} \neq \mathbf{y}_i}{\operatorname{argmax}} \, F(\mathbf{x}_i, \hat{\mathbf{y}}) + \Delta(\mathbf{y}, \hat{\mathbf{y}}). \tag{4.10}$$

Figure 4.3 illustrates some notable stages of the optimization procedure using unbounded loss and a suitable constant C. Here, green indicates ground truth trajectory and red is the negative example. We can observe that, among negative examples generated for a particular training sample, there are examples that reasonably represent possible failure modes. For instance, Figure 4.3(b) shows a trajectory that corresponds to "nothing" i.e. background but it covers a wide range of such mistakes in tracking according to the parameters at that stage. Also, Figure 4.3(c) shows what we call a *distractor* negative example where the system is trying to learn to avoid tracking other people whose appearance is similar to the target pedestrian.

It is worth mentioning that the choice of loss function is crucial in this setting as it implicitly affects the characteristics of the solution of the problem. Specifically, the negative examples i.e. bad trajectories that are generated while solving the optimization problem are a function of loss. It is not straightforward how one can make sure that the negative examples introduced are useful

(a) First iteration: The worst track possible i.e. maximizing the loss only.

(b) Tracking some confusing background "stuff".

(c) Tracking the distractors i.e. another person with similar appearance.

(d) Convergence: tracking the right person.

Figure 4.3: Some stages of optimization: green is the ground truth and red is the negative example.

namely if they introduce representative failure modes in prediction. Moreover, the choice of loss function and the model are closely related. For example, we could modify $\Phi_{\mathcal{F}}(.)$ to account for background clutter captured in the bounding box around the pedestrian to compensate for a shift in the sum of absolute differences when moving from sidewalk to the street and vice versa. Although this can result in a better representation for a trajectory, one can argue that nothing would stop bad trajectories from having the same pattern. So, the learning problem can become more complicated which would not necessarily help with the performance.

4.3 Tracking as Inference

Having estimated the model parameters, the tracking task becomes an inference according to our CRF model namely finding the highest scoring trajectory. Obviously, exhaustive search for the optimal track is computationally intractable. We efficiently solve for this problem using a slightly modified version of the Viterbi algorithm which is given by the following dynamic program

$$M_{(l_C)}^{(t)} = \max_{l_N} \Big(M_{(l_N)}^{(t-1)} + F_{\mathcal{T}}(\hat{\mathbf{y}}^{(t-1)} = l_N, \hat{\mathbf{y}}^{(t)} = l_C) \Big) + F_{\mathcal{O}}(., \hat{\mathbf{y}}^{(t)} = l_C), \tag{4.11}$$

$$t = 2, ..., T, \quad l_N \in \mathcal{N}(l_C),$$

$$M_{(l_{\text{init}})}^{(1)} = 0, \quad \forall l \neq l_{\text{init}}, M_{(l)}^{(1)} = -\infty. \tag{4.12}$$

In fact, back to the probabilistic interpretation, this is equal to finding the maximum of the log posterior given the parameters with the prior of the initial location set to 1 and other pixels set to 0. Each element $M_{(p)}^{(t)}$ corresponds to a pixel and indicates the score of the highest scoring trajectory that originates at the initial location l_{init} and terminates at pixel p at time t. A traceback from the final most scoring location is done to recover the track. In our notation, l_C and l_N refer to the current hypothesized location and its neighboring location(s) respectively. We just search the neighborhood $\mathcal{N}(l_C)$ when trying to find the next possible location instead of doing a full search. Note that this local search is valid since it complies with the nature of the movements of humans as a pedestrian is not expected to jump to a pixel which is far away from the current location. Namely, we are finding an exact solution in the space of "valid" trajectories.

As we will point out later, we need to run our tracker in the original resolution since all the trackers to which we will be comparing our system are doing the same. However, performing inference in high resolutions turns out to be computationally prohibitive even with local search and integral histogram optimizations. Thus, we resort to approximate inference. So, we perform beam search and only consider the N top-scoring trajectories (e.g. $N = 3$ in our experiments) and discard the rest. This allows us to produce the tracking results in the original resolution while keeping the inference feasible. Obviously, beam search will return suboptimal results because it does not explore the whole hypothesis space. However, experimental results show that our approximate inference scheme works well in practice.

Chapter 5

Details and Experiments

We use UBC Fireworks dataset for our experiments [16]. The dataset consists of clips recorded at 1440×1080 resolution using a stationary camera installed on top of a building in downtown Vancouver. Some sample snapshots are shown in figure 5.1. Hence, a top-down view of a moderately crowded scene is captured with variety of moving objects typical to an urban setting present in the image. This includes cars, bikers and pedestrians. The amount of change in illumination, scale and pose is not significant but one needs to deal with background clutter and partial occlusions. The main challenges in the dataset are the presence of occasional crowded blobs of moving pedestrians that introduces many potential distractors and significant background change that occurs when people move from sidewalk to street area and vice versa.

Figure 5.1: Snapshots from the UBC fireworks dataset.

We use 10 manually-labeled trajectories from different sequences for training and 22 other manually-labeled trajectories for evaluating the performance of our system. Both training and test examples are of length 350-500 and contain easy, moderate and hard sequences ranging from a solitary person going through the scene to a pedestrian walking within a crowd with partial occlusions.

5.1 Implementation Details

To reduce training time, we precompute HOG and color histogram distance features prior to training and testing. Appearance templates, however, must be generated online (as they are pairwise potentials) and we compute them efficiently using integral images. We significantly reduce the space of possible trajectories in training by running the Viterbi algorithm in steps of nine pixels in both horizontal and vertical directions so the actual working resolution for Viterbi is 160×120. We define the neighborhood $\mathcal{N}_{(l_C)}$ to be the area within a radius of 2 pixels centered at the current location l_C. This choice is made based upon empirical statistics of the dataset which is in fact a function of the camera angle, average walking speed of pedestrians and frame rate. Similarly, we set the d_{max} in motion model and ρ in bounded loss to the same constant. Note also that because testing processes images at different resolution from training, the motion model obtained from training must be adapted for testing. So, a simple nearest neighbor interpolation of the motion model is performed using the same number of discretization bins as in training and the weights are used directly.

5.2 Automatic Trajectory Extraction

For qualitative evaluation, we design a system that automatically performs the tracking task. Searching for all the targets and learning their appearance as done in [30] is not practical in our problem because we neither can assume a constant appearance nor can we get a reliable segmentation (or pose as in [38]) of them as the view is top-down and pedestrians are far away. As pointed out earlier, we assume that initial locations of the pedestrians are given by a competent detector and the task is to follow them while they are visible. The procedure can be summarized as follows. We divide the whole footage into some fixed length clips and extract the features for each clip. Next, using the HOG-based human detector we locate the people and mark them using bounding boxes in the image. Then, we initialize one tracker per detection for all the detections that are within a specified boundary in the scene and terminate them when they hit some pixel at the boundaries of the region of interest. Since even the state-of-the-art detector misses some instances, we initialize one tracker per

Learning	Loss Type	# CDT	Avg CT	Avg Error
Exact	Δ	15	0.56	11.38
Exact	$\mathbf{\Delta_B}$	**21**	**0.67**	**7.01**
Approximate	Δ	17	0.61	9.90
Approximate	Δ_B	20	0.64	12.24

Table 5.1: Our tracking results with different learning and loss functions on 22 test samples.

detection every 50 frames and perform tracking forward and backward in time and merge the two so we get full trajectories of all the pedestrians that show up in that clip. As a result, we get potentially many trajectories belonging to the same person that differ only in their initial location of detection. Finally, we cluster all the trajectories using the mean-shift clustering algorithm, prune the outliers and false detections and introduce the cluster centers as the final tracking results. Interestingly, this procedure is very successful in practice which was capable of recovering almost all the pedestrians in our experiments.

5.3 Results

We compare the results of our tracking system with the algorithms proposed in [9] and [3]. To gain insight into the importance of the proposed combination of the features, we design experiments with some groups of features turned off. Obviously, we learned different sets of parameters for each combination of the features independently. We used the MIL-tracker software provided by the authors and implemented our own version of [9] which we call the *Collins-Liu* tracker.

We chose 22 challenging trajectories and manually labelled them for quantitative evaluation. As before, we run independent instances of the tracker forward and backward in time in order to get complete trajectories starting from the fixed set of selected detections. Trackers are terminated once they are within a certain number of pixels from the image borders. We use the same procedure to extract trajectories using other methods so we can make a more realistic comparison.

Besides the usual average pixel error measure, we use two other performance measures proposed in [45]. *Correct Detected Track* (CDT) indicates the number of correct trajectories. A track is defined as a CDT if the amount of spatial and temporal overlap with the ground truth exceed thresholds T_{ov} and TR_{ov} respectively, where T_{ov} and TR_{ov} are both set to 0.5 in our experiments. This roughly means that at least half of a CDT must temporally coincide with its ground truth, its length cannot be less than half of its ground truth, and the average spatial overlap must be at least 0.5. *Closeness of Track* (CT) is defined as the average spatial overlap between a ground truth and a

Tracker	# CDT	Avg CT	Avg Error
MMTrack: All	**21**	**0.67**	**7.01**
MMTrack: Hist+Templates	20	0.61	12.74
MMTrack: HOG+Templates	14	0.52	22.24
MMTrack: HOG+Hist	10	0.47	14.40
MILTrack	19	0.61	19.87
Collins-Liu	14	0.54	21.24

Table 5.2: Comparison of tracking results.

system track in the temporally coincident portion of the track. Its value ranges from 0 to 1, with 1 indicating that the track is exactly the same as the ground truth in the temporally coincident section of the track. More detailed explanation of the measures are provided in [45].

We tried exact and approximate learning schemes as well as bounded and unbounded loss while keeping the inference the same for all the experiments. Table 5.1 summarizes the performance of our tracker. As seen in the table, exact training using bounded loss achieves the best result in all measurements among all the configurations of MMTrack. Theoretical guarantees of the optimization algorithm clearly explains the superiority of the exact training over approximate training. Moreover, as motivated earlier by its definition, bounded loss is a better loss function than unbounded loss in all settings as expected. It is well-known that directly optimizing for the measurement of interest is clearly advantageous. Although we are not exactly doing so, bounded loss better matches the nature of our measurements as it roughly mimics the overlap constraint and stops over-penalizing as soon as the overlap becomes zero. Obviously, the quality of the features corresponding to a trajectory does not necessarily become worse if it is shifted away from the ground truth.

The second set of experiments compare our tracker with other trackers on the same test set and experimentally justifies our choice of features. As shown in table 5.2, our tracker outperforms MIL tracker [3] and Collins-Liu tracker [9] in this dataset. One can explain this promising performance by reasoning about the role of different cues in our system. Specifically, HOG feature helps the tracker eliminate areas belonging to non-pedestrian objects, histogram distance maps provide a rough description of the pedestrian and helps alleviate drift whereas appearance templates provide finer levels of the appearance, with the previous frame appearance template allowing some degree of adaptability to appearance change over time. These results, of course, are not directly comparable since the trackers build upon completely different set of features and we just aim at system level comparison. Also, both [9] and [3] are only concerned with appearance modeling and do not include a parameterized motion model.

Interestingly, table 5.2 shows that removing some of the features significantly reduces the performance of our tracker indicating that the combination of HOG, histogram distance and template appearance features is essential in achieving good performance. While each feature group is responsible for avoiding certain types of failures, interactions between groups accounts for difficult situations. Hence. when a feature group is discarded, not only the corresponding failure modes show up but also more complicated failure modes occur as feature groups are not completely independent.

It is worth mentioning that table 5.2 reminds us that the most important components of our tracker are distance histograms and appearance templates. The second row, which corresponds to the system with the second best results, does not include HOG score which implies that our tracker does not really depend on a detector although a good one would slightly improve the performance. Since directly justifying this table might be vague, we would like to use an example to help with understanding the results.

Figure 5.2 is an example that illustrates the importance of our cue combination strategy. As observed in the subfigures, our tracker with only HOG and histogram distance features drifts to a nearby distractor at some parts of the track as it does not know about the initial appearance of the person. The jitter in the track is mostly due to lack of fine details of the appearance which roughly makes the neighbors become equally good according to the model. On the other hand, with HOG and appearance templates, the tracker gets stuck in background area at the boundary between the sidewalk and the street. The reason is that the a significant change in appearance template from frame to frame which occurs at this boundary is rare and hence not supported by the average statistics. So, the tracker is not robust against sudden changes in appearance. Also, since information about average appearance and average background is lacking, drift is inevitable and the role of initial template breaks as background pixels make the stay in sidewalk more rewarding than moving towards the street. The combination of HOG, distance maps and appearance templates manages to track the person correctly. In this case, the system is stable against rapid changes while being reasonably accurate.

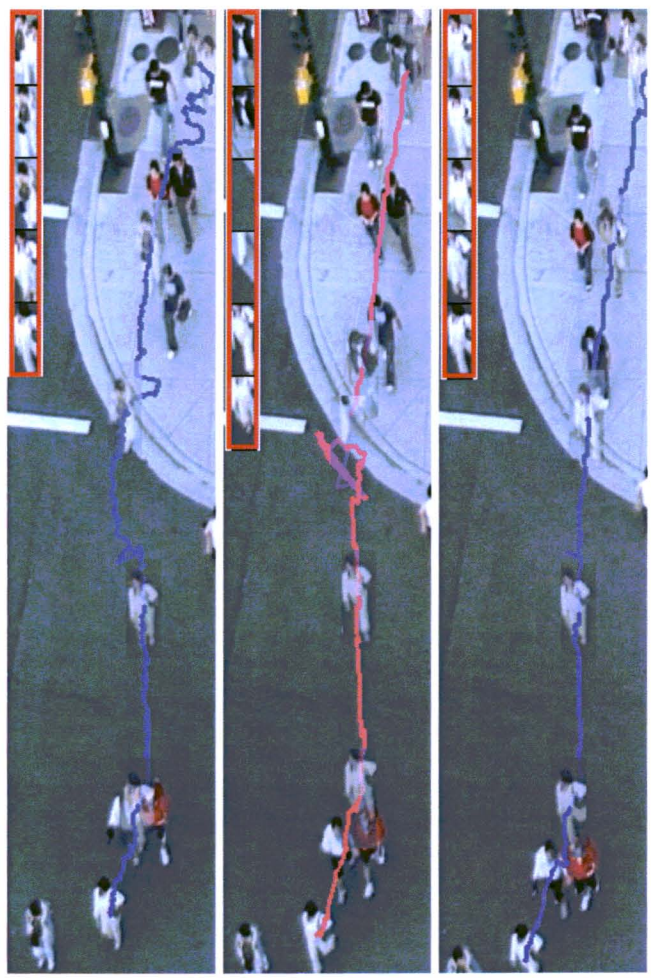

Figure 5.2: An example illustrating the motivations behind our design of features. The tracked "objects" throughout the trajectories are shown in red insets and also superimposed along the trajectories. **Left:** HOG+histograms, **Middle:** HOG+templates, **Right:** HOG+histogram+templates.

Chapter 6

Conclusion

In this thesis, we introduced MMTrack, our offline single target tracking system that employs a large margin learning criterion to effectively combine different trackers. My contribution was adopting learning and inference procedures and designing suitable features for tracking. Although MMTrack is used for pedestrian tracking in this work, we believe that our framework is general and can be used to track other objects too provided that features can reliably describe the target object and handle situations of interest while avoiding confusions for our discriminative classifier. As discussed earlier, even though our best results were obtained when we included a detector signal, we showed that excluding it would not hurt much and we can still perform better than the trackers we compared with. As a result, we have managed to introduce a simple system with few parameters that does not require to model the recognition to do the tracking.

Our tracking system has its limitation in handling severe occlusion and track hijacks caused by significant change in appearance or situations where the background patch is very similar to the appearance of the target. Incorporating mechanisms that would enable single target trackers to explain long-term occlusions while avoiding distractor hijacks turns out to be very challenging. However, a possible future direction is to extend this framework to more complicated systems that can parameterize multi-target tracking. Moreover, learning different parameters for different locations in the image may also be of interest. Such a tracking system would be able to deal with location specific situations that are difficult to handle for a generic tracker. This is motivated by the intuition that the relative importance of the features is likely to be affected by the statistics of background patches and particular occlusions at different locations. On the other hand, designing trackers with more complicated statistics or background models can result in better performance. Finally, defining suitable problem-specific loss functions that directly optimize for benchmark measurements is desirable.

Bibliography

[1] Yasemin Altun, Thomas Hofmann, and Ioannis Tsochantaridis. Large margin methods for structured and interdependent output variables. In Gökhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan, editors, Predicting Structured Data (Neural Information Processing). The MIT Press, 2007.

[2] Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden markov support vector machines. In ICML '03: Proceedings of the 20th international conference on Machine learning, 2003.

[3] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In IEEE Conference on Computer Vision and Pattern Recognition 2009 (CVPR'09), 2009.

[4] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. The Annals of Mathematical Statistics, 1970.

[5] J. Berclaz, F. Fleuret, and P. Fua. Robust people tracking with global trajectory optimization. In IEEE Conference on Computer Vision and Pattern Recognition, 2006.

[6] Stephen Boyd and Lieven Vandenberghe. Convex Optimization. Cambridge University Press, 2004.

[7] Yen-Lu Chow and Richard Schwartz. The n-best algorithm: an efficient procedure for finding top n sentence hypotheses. In HLT '89: Proceedings of the workshop on Speech and Natural Language, 1989.

[8] Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing, 2002.

[9] Robert Collins, Yanxi Liu, and Marius Leordeanu. On-line selection of discriminative tracking features. IEEE Transactions of Pattern Analysis and Machine Intelligence, 2005.

[10] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In IEEE Computer Vision and Pattern Recognition, 2000.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In Machine Learning, 1995.

[12] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. Journal Machine Learning Research, 2002.

[13] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, International Conference on Computer Vision & Pattern Recognition, 2005.

[14] Yves Gr, Johnny Mariéthoz, and Samy Bengio. A probabilistic interpretation of svms with an application to unbalanced classification. In Advances in Neural Information Processing Systems 18, 2005.

[15] Altun Y. Hofmann T., Tsochantaridis I. Learning over structured output spaces via joint kernel functions. In Proceedings of the Sixth Kernel Workshop, 2002.

[16] K. Ismail, T. Sayed, and N. Saunier. Automated collection of pedestrian data using computer vision techniques. In Transportation Research Board Annual Meeting Compendium of Papers, 2009.

[17] K. Ismail, T. Sayed, and N. Saunier. Automated collection of pedestrian data using computer vision techniques. In Transportation Research Board Annual Meeting Compendium of Papers, 2009.

[18] Thorsten Joachims. Learning to align sequences: A maximum-margin approach. In New Algorithms for Macromolecular Simulation. Volume 49 of LNCS, 2003.

[19] Thorsten Joachims. Training linear svms in linear time. In KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006.

[20] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. Machine Learning, 2009.

[21] J. E. Kelley. The cutting plane method for solving convex programs. Journal of the SIAM, 1960.

[22] Bahman Yari Saeed Khanloo, Ferdinand Stefanus, Mani Ranjbar, Ze-Nian Li, Nicolas Saunier, Tarek Sayed, and Greg Mori. Max-margin offline pedestrian tracking with multiple cues. In Seventh Canadian Conference on Computer and Robot Vision (CRV), 2010.

[23] Mathias Kolsch and Matthew Turk. Fast 2d hand tracking with flocks of features and multi-cue integration. In CVPRW '04: Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 10, 2004.

[24] John Lafferty. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In ICML '01: Proceedings of the 18th international conference on Machine learning, 2001.

[25] Christoph H. Lampert and Matthew B. Blaschko. Structured prediction by joint kernel support estimation. Mach. Learn., 2009.

[26] Hong Liu, Lin Zhang, Ze Yu, Hongbin Zha, and Ying Shi. Collaborative mean shift tracking based on multi-cue integration and auxiliary objects. In IEEE International Conference of Image Processing (ICIP), 2007.

[27] S. M. Shahed Nejhum, Jeffrey Ho, and Ming-Hsuan Yang. Visual tracking with histograms and articulating blocks. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, 2008.

[28] Fatih Porikli. Integral histogram: A fast way to extract higtograms in cartesian spaces. International Conference on Computer Vision & Pattern Recognition, 2005.

[29] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In Proceedings of the IEEE, 1989.

[30] Deva Ramanan, David Forsyth, and Kobus Barnard. Building models of animals from video. In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2001.

[31] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. International Journal of Compute Vision, 2008.

[32] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. Neural Computation, 2001.

[33] Lei Song, Rong Zhang, Zhengkai Liu, and Xingxing Chen. Object tracking based on parzen particle filter using multiple cues. Advances in Multimedia Information Processing, 2005.

[34] Martin Spengler and Bernt Schiele. Towards robust multi-cue integration for visual tracking. In Machine Vision and Applications, 2003.

[35] B. Stenger, T. E. Woodley, and R. Cipolla. Learning to track with multiple observers. In IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[36] Ben Taskar. Learning structured prediction models: a large margin approach. PhD thesis, Stanford University, 2005. Adviser-Koller, Daphne.

[37] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In Neural Information Processing (NIPS), 2003.

[38] Leonid Taycher, David Demirdjian, Trevor Darrell, and Gregory Shakhnarovich. Conditional random people: Tracking humans with crfs and grid filters. In CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006.

[39] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In IEEE International Conference on Machine Learning (ICML), 2004.

[40] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, Yasemin Altun, and Yoram Singer. Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research, 2005.

[41] Paul Viola and Michael Jones. Robust real-time object detection. In International Journal of Computer Vision (IJCV), 2001.

[42] Jianyu Wang, Xilin Chen, and Wen Gao. Online selecting discriminative tracking features using particle filter. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[43] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In The European Symposium on Artificial Neural Networks, 1999.

[44] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. ACM Computing Surveys, 2006.

[45] Fei Yin, D. Makris, and S. A. Velastin. Performance evaluation of object tracking algorithms. IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS2007), 2007.