Understanding Virtual Evidence Boosting: A new Derivation and Application to Recognizing Spatial Context with Wearable Cameras

Einführung in Virtual Evidence Boosting und Anwendung auf Visuelle Umgebungserkennung

Diplomarbeit

im Studiengang `Informatik`

angefertigt an University of Washington

und Technischer Universität Darmstadt

von

`Tobias Oberlies`

Seattle, Washington und Darmstadt, Juli 2007

Betreuer: Prof. Dr. Oskar von Stryk

Prof. Dr. Dieter Fox

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Juli 2007 _____

Abstract

Virtual evidence boosting (VEB) is a very promising method for training conditional random fields (CRFs) recently published by Liao et al. [2007]. By applying the principles of LogitBoost, it performs feature selection and parameter estimation for both local and pairwise factors of a CRF in a unified framework and is able to directly integrate continuous observations. While Liao et al. [2007] have shown that the method surpasses other approaches in practical applications, their derivation remains unsatisfactory. In this work we provide a new and detailed derivation, which also includes the foundations of VEB, LogitBoost and the inference method belief propagation. In this context we propose two new versions of VEB which however show inferior performance to Liao's VEB in empirical comparison.

The context for this work is human activity recognition from wearable sensors. We investigate the use of a wearable camera for recognizing spatial context ('inside', 'outside', 'on a bus') and compare color histogram, color-spatial moment, and steerable pyramid image features for this task. The performance of the baseline system, a multi-sensor board integrating seven sensor modalities, can be increased by adding camera features on some of our recordings.

Contents

Chapter 1

Introduction

Non-verbal communication is a key concept in human interaction in everyday life, even beyond its role in human relationships. Without being aware of it, we speak through our actions, and understand other people's state, goals and intentions by observing what they do. This allows us to predict or anticipate a person's next action and if necessary allows us adapt our own actions accordingly. This is for example crucial in traffic, where failing to understand the situation around you can lead to serious accidents.

Recognizing what people are doing around us seems to be a very straightforward, if not trivial task. This mainly due to two facts:

- First of all, we have excellent sensing and perception capabilities. Our eyes can perceive a multitude of relevant details in the human environment and work well across a significant range of lighting conditions. Also, and that's particularly useful for dynamic environments, our visual cortex can reliably detect motion and speeds, even if we are moving ourselves. Where the eyes are limited — they only allow us to see what's in front of us — they are completed by other senses, in particular the ears. They can alert our attention to relevant because potentially dangerous events behind us, and hence allow us to retain an omnidirectional overview of things around us.

- Secondly, we have a huge supply of experience what humans do, either from previous observations of other people or ourselves. From this experience we can deduce the possible goals behind other people's visible actions.

However, if we try to implement the same capabilities in a computer system, the task proves to be very difficult. The discipline concerned with this task is called *human activity recognition*, or in short just activity recognition. It is not a distinct field of research but

has received attention from many research communities, including robotics, machine learning, and ubiquitous computing. Accordingly, the tasks have been investigated are very variable, from tracking people in a room [Pentland, 1996] to inferring the meaning of places to an individual [Liao et al., 2005].

1.1 Taxonomy of Activity Recognition Tasks

To better understand the scope of the activity recognition task pursued in this thesis, it is useful to investigate the different dimensions of activity recognition. Applications can be distinguished according to the setup of sensors with respect to the subject and by the abstraction level of the categories inferred by the system.

1.1.1 Sensor and Application Setup

All activity recognition applications have in common is that they try to capture and infer information about people in the real world. They differ however in where the sensors are placed which respect to the people.

Early activity recognition placed the sensor networks into a room or building to make them "smart rooms" [Pentland, 1996], which unlike other computer systems are able to perceive their user's activities and so can be more "helpful". Similar systems have emerged for mobile robots allowing the agent to recognize the movement of people around it [Schulz et al., 2003a]. This capability is also sought after in traffic where a partially autonomous sensor system in a car can help to prevent collisions with pedestrians [Gandhi and Trivedi, 2006]. A future application which can also draw significant benefits from activity recognition are mobile service or assistance robots. These will need to understand spoken commands as well as non-verbal communication and be able to safely navigate around natural and dynamic environments. All these applications have in common that people and their activities are detected from remote sensors, like range finders or cameras. In particular the latter can provide a rich set of information for remote recognition of human activities. Indeed, "looking at people" has spun large interest in the computer vision community, but due to the high complexity of the task — discovery and tracking of a possibly non-rigid body (because of clothes) with tens of degrees of freedom in a variable environment — no robust, general purpose solutions have emerged yet [Gavrila, 1999].

A different class of very successful applications of activity recognition use *wearable sensors*, i. e. sensors that are carried around by a person, to infer information about the wearer. This is a main focus of the ubiquitous computing community and has attracted a

lot of attention in recent years. It allows for other types of sensors, e. g. accelerometers or GPS, which provide much more concise data than remote sensing systems like cameras. The applications are for example mobile devices like PDAs which provide relevant information at the right time reducing the perceived burden of interruptions [Ho and Intille, 2005]. A more elaborate level of activity recognition is required by the system called *Opportunity Knocks*, which helps cognitively-impaired people to use public transportation independently by making them aware of deviations of the usual route and helping them recover [Patterson et al., 2004]. These two examples have in common that they give live feedback to the users. In other cases the wearable devices only record daily activities for off-line evaluation. Applications range from long-term health monitoring for the elderly [Philipose et al., 2004] and social science studies [Wyatt et al., 2007] to automatic mission summaries for soldiers (see section 1.2).

1.1.2 Activity Levels

Most commonly, the output of an activity recognition system is a label for what the individual wearing the sensors or the observed people are doing at a particular point in time. These labels can either be triggered by the actions themselves or be reported at a constant rate. The set of labels which are distinguished strongly depend on the application, but we can categorized them by their level of abstraction.

At the bottom level is the immediate action performed by a person, like walking, speaking, or handling a certain object. The middle level of abstraction captures more complex, compound actions, like crossing the street, talking with a friend, or making dinner. These activities are obviously harder to detect or infer, but they contain a much more valuable information, which for example allows a car's collision avoidance assistant to hit the brakes, a mobile phone to not suppress a call (because it's a casual conversation), or a caring person to see that their elderly relative is still capable to manage her everyday life. The highest level of abstraction involves the goals and intentions of people, like the walking towards a target or being on the trip home. The goals may span a few seconds up to several hours, and therefore are particularly important to predict the low or middle level action a person will perform next [Liao et al., 2004].

1.1.3 Location in Activity Recognition

Especially for the higher abstraction levels of activity, the location of the wearer is often a cue for the activity performed. Consider the case that a person is 'at a supermarket', which almost certainly means that he is buying things, unless he is 'at work'. The concept

"location" can be understood in very different ways in activity recognition applications, so we can distinguish the following three notions:

1. An obvious interpretation is the absolute position in terms of latitude and longitude. Determining the position of the person carrying a mobile device is called *localization*, and it is a basic layer in many activity recognition systems. The satellite-based *global positioning system* (GPS), which allows to directly measure these values, has been found unreliable in human environments, so some solutions rely on other radio beacons like GSM and WiFi antennas instead [LaMarca et al., 2005, Ferris et al., 2006]. The absolute position can be used to find a users *significant places*, which is usually simply defined as the places a person stays at for a certain time [Ashbrook and Starner, 2003, Marmasse and Schmandt, 2002]. These systems identify distinct places, but they do not automatically assign any semantic labels to these places (cf. item 3 in this list).

2. The objective category of an environment, which we call *spatial context*, is a further notion of location. It distinguishes labels on the level of abstraction of inside and outside [Subramanya et al., 2006], or certain types of rooms like office, kitchen, corridor etc. [Torralba et al., 2003].

3. The *place meaning* describes the role that a place has for a particular person, like home, work, or home of a friend [Liao et al., 2005, Hightower, 2003]. These labels are specializations of the spatial context labels in the sense that 'my office' applies to less places than the category 'office'.

The process of finding the label from the latter two cases is both called *place classification* or *place labeling*, so throughout this work we will resort to the unique expression of "recognizing spatial context". This will be the focus of the activity recognition task pursued in this work.

1.2 Motivation for Recognizing Spatial Context

Spatial context has only rarely been detected in activity recognition systems (see related work in section 2.3). However we think that spatial context is in fact a very important notion of location because it captures general concepts that can be applied to arbitrary locations and not only certain limited areas or a few places. Applications based on spatial context labels can therefore work everywhere. For example 'sidewalk' and 'building entrances' are also spatial context labels, and detecting these could be the key to a

successful pedestrian target predictor. While this is a hard challenge for future research, even much more simple labels are useful for many applications. In this work a wearable sensor system will be used to classify the spatial context into 'inside' and 'outside', and 'on a bus'.

This information may by itself be interesting. The *experience sampling method* is used in social science studies to ask people about their everyday lives. In order to reduce recall errors, experience sampling uses a portable device that prompts the user to answer a questionnaire in place. Traditionally this is done in regular intervals, but the approach can be made more efficient by only prompting in a certain spatial context or after particular activities [Intille et al., 2003]. In a different application, Torralba et al. [2003] use the spatial context information inferred by a camera-based system to get a prior for which other objects may also be visible in the image.

Furthermore detecting spatial context can be an important layer in a larger activity recognition system. Applications which identify significant places and their meaning to a user [Liao et al., 2005] could use the spatial context label as additional information since for example only few significant places can be outside. In other cases, detecting the locations 'car' and 'bus' could have helped to differentiate the corresponding modes of movement, which has proven to be very hard if only GPS traces are available [Liao et al., 2004].

Finally, it can't be dismissed that recognizing spatial context has a military application. The preceding work by Subramanya et al. [2006] was funded under the DARPA (Defense Advanced Research Projects Agency) program *ASSIST* (Advanced Soldier Sensor Information System and Technology). The objective of this program is to develop sensor systems to improve the soldiers 'after mission' reports. Creating these from memory only has proven to be difficult for the soldiers, and hence information essential for the military mission is lost. The mission reports are supposedly particularly important for urban combat environments [DARPA, 2004].

As a component of a future automatic report generator, it may useful to to augment the absolute positions with a list of the buildings that have been visited by the soldiers. In spite of the reasonable assumption of known bounding boxes of all buildings in the visited area, it is not possible to reliably infer this information from GPS. A recording from the DARPA evaluation mission in Aberdeen, Washington shows the reported GPS position meandering outside and over neighboring buildings while the individual is actually resting inside another building. Only by explicitly and jointly detecting the spatial context indoors, outdoors or vehicle, the presence in buildings can be reliably inferred.

1.3 Challenges of Activity Recognition

Parallel to the human cognition, an artificially intelligent activity recognition systems consist of several layers, each with different tasks and challenges.

1. The first task is the design of the sensors, which should provide rich and meaningful data, while being non-intrusive and inexpensive. In many cases one can resort to standard integrated hardware. A notable exception to this are wearable RFID readers like the *iBracelet* [Smith et al., 2005] which detect the RFID tags of grasped objects.

2. The data collection is followed by a low level processing step called *feature extraction*. In this step the raw data is refined to make it more concise and so allows to learn principled dependencies with hidden labels more easily. It involves general data analysis and dimension reduction techniques — for example principal component analysis or the Fourier transformation — and specifically designed feature functions that extract higher level information that is known to be contained in the raw data.

3. The model defines the scaffolding for the logic dependencies between features and labels that can be learned. The design of the model has to take into account how different label variables, for example low-level and high-level activities, depend on each other which may yield complex hierarchies [Liao et al., 2004].

4. Closely related to the model is the choice or design of the learning algorithm and inference mechanism. Real-world activity models are too complex as if they could by created by hand. Therefore a machine learning algorithm is used to determine the parameters of the model from examples, i. e. data recorded using the sensors and low-level processing. Once the model is fully determined, an inference mechanism is required to deduce the most likely activity or probabilities of activities. Especially for online applications, like context-aware information systems, the efficiency of that method is also important.

In the design of a solution all of these have to be considered, but in this work the main contributions lie in the second and fourth layer of activity recognition.

1.4 Contributions

The main focus of this work is on a recently published method for supervised learning of relational data (cf. definition in section 3.2) called *virtual evidence boosting* [Liao et al., 2007]. This method is suitable solution for the fourth task in an activity recognition system but also widely applicable to many other problems. Furthermore, for recognizing spatial context different sensors and features are evaluated in an empirical comparison. In detail, the contributions are as follows:

- A complete and detailed re-derivation of virtual evidence boosting (VEB) for training conditional random fields is developed in chapter 5.4 which provides new insight about the underlying optimization strategy of the algorithm. The derivation strictly follows the ideas in the original publications [Liao, 2006, Liao et al., 2007] but due to an error in their work yields an in parts different method (see section 5.4.6). Interestingly, empirical comparison shows that Liao's VEB still surpasses our versions *VEB with neighborhood-based Newton steps* and *VEB with piecewise Newton steps* in the context of our activity recognition application (see section 6.3.2).

- VEB is based on the learning method LogitBoost by Friedman et al. [2000]. A formal derivation for this method is given in chapter 4 providing the background for the introduction to VEB. We also close a gap in the original proof, where the optimality of the update steps remains vague (see section 4.2).

- The convenient notation of conditional random fields allows for a new informal proof of the correctness of *belief propagation* in loop-free graphs. It avoids the fuzzy concept of the "beliefs" [Yedidia et al., 2002] but bases the explanation on the formula content of each message. Together with an introduction for readers who are not yet familiar with the inference method belief propagation, the correctness will be illustrated in section 5.3.

- Finally, we investigate the use of a web-cam in a wearable activity recognition setup to recognize the spatial context categories 'inside', 'outside', and 'on a bus' (chapter 6). We evaluate color histogram, color-spatial moment, and steerable pyramid image features and find color features or a combination of all features to be most successful. The camera sensor is however outperformed by the *multi-sensor board* [Lester et al., 2005], combining sensors for light, audio, temperature and others. Still, the accuracy of label predictions can be improved by adding camera features on our datasets which include traces recorded at night.

Chapter 2

Related Work

In this section we discuss previous publications which are related to the topics of this thesis. The first two sections present related learning methods, followed by three sections on work related to the specific task of recognizing spatial context from camera data.

2.1 Boosting

Boosting algorithms have been hugely popular in recent years. Its key idea is to boost the performance of a classifier by training and combining an ensemble of instances. The first practical version was *AdaBoost*, published by Freund and Schapire [1997]. Its roots lie in the *probably approximately correct* (PAC) learning theory which defines two types of learning algorithms: a *strong learning algorithm* has to be able to classify all but a small fraction of the samples correctly (with a given probability). However the class of functions for which those strong learners provably exist is severely limited [Valiant, 1984]. A *weak learner* is only required to classify slightly better than random guessing and hence exists for a much wider class of functions [Kearns et al., 1995]. Surprisingly, given enough data, any weak learner can be boosted into a strong learner [Kearns and Valiant, 1994]. Schapire [1990] provided the first polynomial time algorithm to do so, and a simplified version, called *Boost by Majority*, was published by Freund [1995]. The remaining limitation of the latter was however that each of the weak classifiers was expected to yield the same performance, which is usually not true. AdaBoost combined Boost by Majority with the *Weighted Majority Algorithm* by Littlestone and Warmuth [1994], adaptively weighting each of the weak learners according to their performance (hence the name *Ada*ptive *Boost*ing). An important generalization to AdaBoost was presented by Schapire and Singer [1999]. It allowed the weak learner to output a real-

valued confidence level instead of discrete class labels.

AdaBoost yielded a surprisingly good generalization performance, but it was not well understood why. Schapire et al. [1998] observed that AdaBoost often keeps decreasing the generalization error even when the training error has already reached zero. The driving force seemed to be to increase the classification margin. Also, they proved a bound on the generalization error depending on the margin achieved over the test set. Therefore they concluded that maximizing the margins is the key to a low generalization error. Breiman [1999], however, showed that this is not entirely accurate: His explicit max-margin algorithm named *arc-gv* (adaptive reweighting and combining – game value) achieves uniformly higher margins, but has higher generalization errors. An important step towards understanding AdaBoost was made, when it was described in terms of optimization: it is a gradient descent method optimizing an exponential criterion [Friedman et al., 2000, Frean and Downs, 1998]. This criterion can be seen as an approximation for maximum likelihood, which is a common modeling criterion in the statistical community, and explains in part the good performance of AdaBoost. Having found this insight, Friedman, Hastie, and Tibshirani [2000] developed LogitBoost, which optimizes the data likelihood directly (cf. section 4). The optimization view also allowed for a principled comparison of the various flavors of boosting [Mason et al., 2000].

Boosting can also be applied to multiclass learning problems. The approach of simply combining weak learners for multiple classes (*AdaBoost.M1* by Freund and Schapire [1997]) may not practical because the weak learners don't necessarily achieve a correct classification rate of 0.5 or better. More successful algorithms reduce the multiclass problem into several binary problems by extending and transforming the training data set (e. g. *AdaBoost.MH*, presented in [Schapire and Singer, 1999]) or by using multiple predictors for one-vs-reference (*AdaBoost.M2* [Freund and Schapire, 1997]) or one-vs-all classification (*LogitBoost (J classes)* [Friedman et al., 2000], see section 4.4).

An extensive introduction to Boosting and its theoretic properties can be found in Meir and Rätsch [2003].

2.2 Probabilistic Learning for Activity Recognition

Boosting is not the most suitable method for an activity recognition (AR) task because it can only predict scalar labels and hence has to assume that the label sequence consists of independent instances. Still, boosting has been used in some applications [Bao and Intille, 2004], but other experiments show that the recognition accuracy can be improved by exploiting dependencies [e. g. Lester et al., 2005, Subramanya et al., 2006].

Therefore a learning method is required which finds a predictor for vectors of dependent labels. The approach of probabilistic learning methods to this task is to learn a probability distribution over inputs $\mathbf{X}$ and labels $\mathbf{Y}$ because this allows to infer the most likely labels for a given input. Distributions over multiple random variables are typically represented by *graphical models*, which factorize the distribution according to a graph. Graphical models come in various flavors: as *directed graphical models* (i. e. the edges are directed) like *Bayesian networks* [Korb and Nicholson, 2004, Pearl, 1988] or as *undirected graphical models* like *Markov random fields* [Kindermann and Snell, 1980]. A particularly intuitive graph representation for undirected graphical models are the so-called *factor graphs* [Kschischang et al., 2001] (cf. section 5.2), which therefore have replaced the equivalent clique-based graphs in recent publications [e. g. Sutton and McCallum, 2006, Yedidia et al., 2005]. Many activity recognition tasks use graphical models which are specialized versions of the two main flavors. Bayesian Networks for example which consist of a connected sequence of *time slices*, i. e. copies of a set of variables representing the state at a particular point in time, are called *Dynamic Bayesian Networks* [Dean and Kanazawa, 1988]. Also, the well-known *hidden Markov models* (HMM) [e. g. Rabiner, 1989] can be considered to be a special case of these where each time slice only contains one hidden and one observed variable.

Graphical models can be trained to either represent a joint probability distribution over the observations and hidden labels $\mathrm{P}(\mathbf{X}, \mathbf{Y})$ (*generative models*) or a conditional distribution $\mathrm{P}(\mathbf{Y} \mid \mathbf{X})$ (*discriminative models*). The latter are particularly suitable for classification problems because the values of the observed variables are always given anyway and hence the model instance doesn't need to contain any information about the general distribution of these values. The learner can therefore make best use of the available degrees of freedom to represent the conditional distribution we are interested in [Minka, 2005]. Generative models on the other hand implicitly contain the marginal distributions over all observations $\mathrm{P}(\mathbf{X})$, which can have very complex dependencies that are intractable to represent correctly [Sutton and McCallum, 2006].

The conditional equivalent of Markov random fields are the so-called *conditional random fields* (CRFs) [Lafferty et al., 2001]. Although CRFs have been originally used as linear chains, they also generalize to arbitrary topologies. Especially if large datasets are used, which is often the case in activity recognition (cf. section 6.2), training these CRF is a challenging problem. The baseline approach is to maximize the likelihood with any standard gradient descent optimization method (ML training) [Sutton and McCallum, 2006]. However the problem with that approach is that it requires to run inference for every gradient evaluation, which can be very expensive even for approximate inference

algorithms [Liao, 2006]. A possible solution to this is to use other optimization criteria which allow for faster training. A classic approximation is maximum pseudo-likelihood (MPL), [Besag, 1975] in which all neighbor variables are treated as observed and so no inference is required. The same concept of learning small parts of the graph independently can be applied to even smaller parts, namely a single edge rather than a full neighborhood [Sutton and McCallum, 2005]. While still requiring inference, *virtual evidence boosting* [Liao et al., 2007, Liao, 2006] approximates the optimization criterion so that the second derivative can be computed and hence a very low number of iterations is required (see also section 5.4).

A special requirement of activity recognition for the learning method is that it needs to be able to handle continuous observations $\mathbf{X}$ and select relevant features from a possibly very large feature set. Neither of these requirements is met by ML training of conditional random fields. A workaround which has been used in AR applications is to train decision stumps using boosting and then use the results as input to a graphical model for learning label dependencies [Lester et al., 2005, Subramanya et al., 2006]. A more successful solution is provided by virtual evidence boosting [Liao et al., 2007] which combines both these steps in a unified manner (cf. section 5.4). A similar approach called *boosted random fields* was developed for a vision task by Torralba et al. [2004]. This method however makes several approximations which may not be valid in activity recognition and is therefore generally surpassed by virtual evidence boosting [Liao et al., 2007].

2.3 Recognizing Spatial Context

There has been little previous work which specifically covered the activity recognition task of recognizing spatial context from wearable camera. However there are many related topics from activity recognition, ubiquitous computing, and image processing research which cover parts of our task. This section presents solutions to recognizing spatial context from various sensor systems and a notable related robotics application. The following section 2.4 introduces work on localization and place recognition and discusses a possible application to context recognition. Finally a survey of scene classification approaches in section 2.5 provides features for the integration of image data into a context recognition system.

The activity recognition task pursued in this work is based in part on the work of Subramanya, Raj, Bilmes, and Fox [2006]. They estimate low-level activities, absolute positions, and spatial context (indoor, outdoor, in a car) from multiple sensors, not including a camera. Except for a global positioning system (GPS) receiver, all these

sensors are united on a multi-sensor board, which had been previously used by Lester et al. [2005]. Both approaches use the same features and boosted decision stumps to feed into a hidden Markov model, respectively a complex dynamic Bayesian network. Liao et al. [2007] showed that accuracies of the estimated activities and spatial context can be improved by using virtual evidence boosting. Using the multi-sensor board and the feature set designed for it will be one of the baseline approaches for the experiments in chapter 6.

The second closely related work was published by Torralba, Murphy, Freeman, and Rubin [2003]. From a helmet mounted web cam, they recognize 17 environment categories, including office, conference room, corridor, street, and plaza. The features are based on the response to a steerable filter bank [Simoncelli and Freeman, 1995] averaged over 16 image subblocks. For a forced classification, i. e. not allowing a don't know classification if the model is undecided, they achieve a precision of 45%. They also report this to be much better than color-based classification, although with a very primitive color feature, at a correct classification rate not exceeding random guessing. The steerable pyramid feature set will be the second baseline approach for for inferring spatial context (see sections 6.1.2 and 6.3.1).

Motivated from robot mapping applications, Posner et al. [2006] developed an algorithm for clustering the image series captured from a mobile robotic by their similarity. The distance measure used is based on presence of visual words (cf. section 2.5). While the clusters sometimes capture scene concepts, they are mostly specific to certain places or dominant objects. However this unsupervised method could be interesting as part of a learning approach with partial labels.

2.4 Localization and Place Labeling

There are two related fields of research concerned about other notions of location (cf. section 1.1.3), which have attracted more attention in recent years. It may be possible to apply solutions found in this areas to recognizing spatial context, however there are several shortcomings.

Due to the unreliability of GPS in urban environments — GPS often fails inside buildings or near tall buildings [LaMarca et al., 2005] — other systems have been developed to determine the absolute position of mobile devices. They use the existing GSM or WiFi beacons to determine the location through triangulation and signal strength. The solutions either require that beacon locations are known [LaMarca et al., 2005], or that training data which includes the true locations is collected [Ferris et al., 2006, Cheng

et al., 2005]. If a detailed map with individual buildings is available, the absolute position information may be used to infer the spatial context class 'inside'. However, the accuracy is critical, ruling out GSM-only based systems with a median error of 20–30 m [LaMarca et al., 2005] and challenging GPS (see section 1.2). A spatial context category like 'in a vehicle' can be very well be determined from position data, but a distinction between bus and car for example is again very difficult [Liao et al., 2004].

A theoretic alternative for fine-grained spatial context recognition from absolute position is using a systems which deploys dedicated beacons or sensors to enable localization [Hightower and Borriello, 2001, Schulz et al., 2003b]. They provide excellent accuracies, but installation cost make them unpractical for large scale applications.

Another field of active research is recognizing and automatic labeling of significant places of users [Liao et al., 2005, Hightower et al., 2005]. Places are found by their absolute positions or by the signature extracted from nearby wireless beacons and then assigned the role it serves for the person carrying the mobile device. This label often bears information about the spatial context class, for example a label at home would allow to infer quite reliably that that place is inside. However, inferring the place meaning is the harder problem. In fact, having spatial context labels could be a significant help for this task.

2.5 Scene Classification

While cameras have been rarely used as worn sensors for activity recognition tasks, there has been a lot of research in classifying the overall scene (as opposed to individual objects) depicted in images. This is often motivated by the insight that semantic categories would allow significant improvements in *image retrieval* systems, which are search engines for images. The number of classes distinguished range from two (inside/outside) [Szummer and Picard, 1998] to 14 classes [Fei-Fei and Perona, 2005], including living room, bedroom, kitchen, office, inside city, suburb, forest, and open country. These labels precisely match the spatial context notion of location which we are interested in in this work. Unlike here, however, the images are photographs of good quality and fairly distinctive and representative (cf. section 6.2). Nevertheless scene classification research has provided interesting approaches and a rich set of useful image features. These will be examined after the presentation of those scene classification applications which include the distinction indoor/outdoor.

For precisely this task Yiu [1996] uses a joint color histogram with bins generated by clustering and a predominant texture direction feature [Gorkani and Picard, 1994]. With

her best classifier she achieves an accuracy of 92% on a small set of images. Szummer and Picard [1998] find that their results are improved by computing features for 16 subblocks of the image and hence capturing spatial and local information. With color channel histograms from a rotated RGB space and MSAR texture features [Mao and Jain, 1992], they achieve a correct indoor/outdoor classification rate of 90.3%. For a hirarchical classification approach, Vailaya et al. [2001] distinguish 'indoor', 'outdoor', and 'close up' images for a further subdivision into eleven classes. For the 3-class distinction layer, they report an accuracy of 90.6% and find color moments on sub-images to be the most suitable features (cf. section 6.1.4 (ii)). They also use a multitude of other features, including color histograms, color coherence vectors, edge direction histograms, and texture filters.

Image features can be generally categorized into *global features*, e. g. histograms, and *local features* like texture filters. Local features are specific to individual pixels, i. e. there is one value for every pixel, and are therefore often averaged over parts of the image with only the averages being used as feature values. Popular partitions include regular grids of rectangles and the split into center and four corners [e. g. Stricker and Dimai, 1996]. Global features may also be made more location specific by computing feature values for parts of the image. The following lists common global features and image filters.

Histograms. For a histogram, each pixel is mapped into a bin according to its color and then the number of pixels in each bin is counted. The resulting vector is typically normalized by dividing through the total number of pixels in the image. While the bin setup is straightforward for grayscale images, there are multiple possible generalizations for the use of color.

In a *joint histogram* each bin is simply a three-dimensional subset of the 3D color space (e. g. [Swain and Ballard, 1991]). A common approach is to split each color axis into N ranges, which results in N^3 bins. Therefore a distinctive partition may consist of a very large number of bins. As a remedy, a non-uniform segmentation in the HSV color space [Smith, 1987] has been proposed [Lei et al., 1999] (see also section 6.1.3). Another approach to this problem is to generate the bins automatically by a clustering method. Although this may yield bins focusing on the significant areas of the color space, it is questionable if this can be achieved with k-means clustering, the approach pursued in [Yiu, 1996, Vailaya et al., 2001].

Alternatively, histograms of color images can be created by simply concatenating several 1D histogram. The standard approach for this is to create one histogram per color channel [Szummer and Picard, 1998]. This makes the choice of color space very important because information is lost through the projections. Szummer and

Picard [1998] propose the use of the *Ohta color space* [Ohta et al., 1980] which rotates the RGB space so that the axes approximately match the directions of the largest eigenvectors of the distribution of colors in natural images. Sural et al. [2002] separate pixels into hue dominance (saturated colors) and intensity dominance (unsaturated colors and grayscales) and create a 1D histogram for each subset.

Color coherence vectors are an extension of color histograms. The feature has been proposed by Pass et al. [1996] in order to capture some local context within a histogram. A pixel is defined *coherent* if it is part of a connected region of similarly colored pixels. A separate color histogram is produced for coherent and non-coherent pixels. This concept can be further generalized by adding local properties as new dimensions to the color space [Pass and Zabih, 1999].

Edge direction histograms are one-dimensional histograms counting the frequency of edge orientations [Tamura et al., 1978]. The edges can be detected with any standard edge detection algorithm, for example a Canny edge detector. In order to detect straight edges, supposedly specific to man-made environments, local coherence information was included by Vailaya et al. [1998].

Dominant directions. Picard and Gorkani [1994] combines the results from steerable filters at different scales to extract one or more dominant directions in the image. The feature value consists of one or more edge directions with the respective strength. This is a more compact representation than a full histogram and hence is suitable for repeated computation over sub-images [Yiu, 1996]. Optimizations for natural scenes are proposed in [Gorkani and Picard, 1994].

Color-spatial moments. Stricker and Orengo [1995] introduce the idea to approximately capture the distribution of colors in the image by the distribution's central moments. The moments are computed separately for every color channel, which is equivalent to a diagonal approximation for the 2^{nd} and higher order moments. Aiming to also capture the spatial distribution of colors in the image, Stricker and Dimai [1996] divide the image into parts and compute the color moments for each of them (cf. section 6.1.4 (ii)). The inverse approach is pursued by Lei et al. [1999]: They discretize the colors with a histogram and then compute the moments of the pixel position distribution for each histogram bins (cf. section 6.1.4 (i)). We empirically compare these types of color-spatial moment features and present the results in section 6.3.1.

Texture features. Textures are regular patterns of grayscale values. In order to segment images composed of several textures, many different statistical, geometrical, structural, model-based and signal-processing features have been developed (surveys and comparative studies can be found in [Gool et al., 1985, Reed and du Buf, 1993, Randen and Husøy, 1999, Singh and Singh, 2002]). Texture features have also been successfully used in the natural scene classification context. Popular choices include *steerable pyramid filters* which detect edges at different scales and orientations [Simoncelli and Freeman, 1995] (see also section 6.1.2) and *multiresolution simultaneous autoregressive models* (MSAR) [Mao and Jain, 1992], where the feature value is the difference to a model generated from neighboring pixels.

Very interesting recent approaches in scene classification use new statistical techniques from text modeling, namely *probabilistic latent semantic analysis* (pLSA) [Hofmann, 1999] and *latent Dirichlet allocation* (LDA) [Blei et al., 2003]. The equivalent of words in images are the so-called *textons* or *visual words*, which are clusters of local feature vectors automatically created by K-means clustering [Leung and Malik, 1999, Sivic and Zisserman, 2003]. In modern approaches the clustering is based on local descriptors like SIFT (Scale Invariant Feature Transform) [Lowe, 1999, 2004] or similar approaches [Mikolajczyk and Schmid, 2005, Bay et al., 2006]. The number of textons can be either fixed if they are computed on a fixed grid or variable by using a detector for *salient regions* [Kadir and Brady, 2001, Lowe, 1999, Mikolajczyk et al., 2005]. The statistical model then describes the images as a mixture of aspects, which in turn are multinomial distributions of textons.

pLSA or LDA can be used to automatically extract these aspects. The resulting aspect distributions for each image can be used as input for a classification algorithm [Quelhas et al., 2005]. In this case pLSA or LDA act as a statistical clustering method and is used with the aim to automatically extract meaningful concepts like sky, rocks, foliage, etc. which had to be labeled by hand in other approaches [Oliva and Torralba, 2001, Vogel and Schiele, 2004]. Quelhas et al. [2005] successfully classifies 92.2% of indoor/outdoor images and further states that the result only deteriorates to 88.6% if a mere 2.5% of the training images are labeled.

LDA can also be used to create a full generative image model, which is the probability distribution of textons for any image from a set of images. By learning a LDA model for each scene category, Fei-Fei and Perona [2005] classify new images by choosing the category which best explains images' distributions of textons and achieve a respectable accuracy of 65% for a joint classification of 14 (!) classes. (Some of these classes were listed a the top of this section.)

Chapter 3

Foundations

This chapter will introduce the notational concepts used throughout this thesis, in particular in the derivations of the probabilistic learning methods LogitBoost and virtual evidence boosting. The second section formally defines the task which is solved by both methods, a *supervised learning problem*.

3.1 Notation

Most of the notation follows the common convention, with the exception of the square bracket index alternative and subscripts in angle brackets. Still further definitions of mathematical operators and concepts are also given here to avoid misunderstandings. The meaning of variables and indices as well as further special purpose symbols are defined near their first uses.

Vectors are distinguished from scalar values by boldface, for example $\mathbf{y}$. Vector entries are identified by a subscript which identifies the element, i. e. y_t is the t-th element of label vector $\mathbf{y}$.

Indexing. In line with the vector entry definition, we will use subscripts to index the elements of a compound structure, or the other way around omit a subscript to denote the compound of all indexed entities. So $\mathbf{x}$ stands for the "vector" of all observation vectors $\mathbf{x}_t$ and $F_{A,j}$ for a particular one of the model functions F. Where the multiple indexing is necessary or the subscript is already used for other purposes (which are unambiguously identifiable as such), some indices may be placed in square brackets behind the symbol. Belief propagation messages $m_{A\to 1}$ for example are vectors, so to select the j-th entry of that vector, we write $m_{A\to 1}[j]$.

Selecting multiple vector elements. A subscript in angle brackets like $\mathbf{y}_{\langle A \rangle}$ describes the vector of all entries of $\mathbf{y}$ which correspond to the elements of the set A. These are those entries with matching indices, and in few cases also the symbols have to correspond. For example if $A = \{Y_t, Y_{t-1}, \mathbf{X}_t\}$, then $\mathbf{y}_{\langle A \rangle}$ stands for $(y_t \; y_{t-1})^T$, or $F_{\langle \Psi \rangle}$ is the vector with an entry F_B for every $\psi_B \in \Psi$. While this non-standard notation may seem a little fuzzy, it is still very powerful and should become clear in the application. Note however that the factors ψ are indexed by sets, e. g. $\psi_{\{Y_t, Y_{t-1}\}}$, which simply identifies them and is not an instance of the angle bracket subscript notation.

Sums and products have two standard ways of specifying their ranges: through lower and upper bounds $\sum_{j=1}^{J}$ or set membership expressions $\sum_{\psi \in \Psi}$. We also use a third notation which only gives a variable like $\sum_{\mathbf{y}}$ and denotes the sum over all values of $\mathbf{y}$. The domain of the variables, which would for the instantiation $\mathbf{y}$ be the domain 'dom($\mathbf{Y}$)' of the vector of random variables $\mathbf{Y}$, becomes clear in the context.

Stochastic notation. As indicated in the previous example, we use uppercase roman letters for random variables and the respective lowercase letter for instantiations. Conditional probabilities are written as $\mathrm{P}(Y = y \,|\, \mathbf{X} = \mathbf{x})$ and the random variable may be omitted where the result is unique as in the case of $\mathrm{P}(y \,|\, \mathbf{x})$. Learning algorithms often represent conditional probabilities using function parameterization. To make the parameter explicit, it may be added on the conditional side, separated by a semicolon, e. g. $\mathrm{P}(y \,|\, \mathbf{x}; F)$.

Other functions and symbols The *indicator function* or *Boolean function* $\mathbf{1}_{\langle j=k \rangle}$ is equal to one if and only if its argument (again in angle brackets) is true, and zero otherwise. A pair of vertical lines can both denote the set size $|A|$ or the absolute value $|f|$. A single, long vertical line at the end of a term like $\alpha[y_t]\big|_{y_t=j}$ denotes a parameter substitution, i. e. the example term is equivalent to $\alpha[j]$. This is often used in the context of parameter updates, e. g.

$$\frac{\partial\, \ell(F+f)}{\partial f}\Big|_{f \equiv 0}$$

sets the function f to constant zero in the derivative.

3.2 Supervised Learning Problems

Both probabilistic methods that will be illustrated in this work are algorithms for *supervised learning problems*, and most activity recognition tasks are set up as problems from this class. It begins with some kind of information y that we would like to know in a computer system, like the activity performed by an individual carrying a mobile device. However this information cannot be measured directly but only some other data which is correlated with the hidden information. This data is called the *observations*, *feature values*, or simply *input values* and denoted by the letter $\mathbf{x}$. We would like to find a *prediction function* $H(\mathbf{x})$ which can predict the correct label y given the observations $\mathbf{x}$. So to learn this function, a learning algorithm is presented with examples pairs of observations $\mathbf{x}^{(s)}$ and the respective correct or *true labels* $y^{(s)}$.

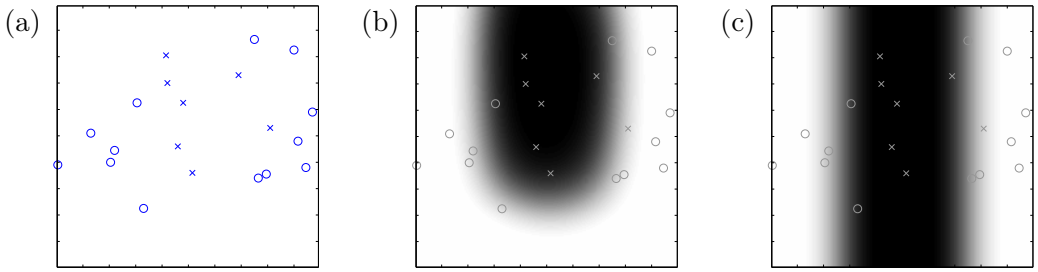

Figure 3.1: (a) A sample set with input data $\mathbf{x} \in \mathbb{R}^2$ and binary labels. The points could be samples of the distribution (b), but they have actually been generated with the distribution (c). The shade shows $\mathrm{P}(y \mid \mathbf{x})$, $\mathrm{P}(\mathbf{x})$ is not shown.

We obviously want the prediction function to not only return the labels for those observations that were part of the examples but also for other previously unseen observations. This is possible to a certain extend because there is some kind of pattern behind the observations and the labels. In statistics, these patterns are described by a distribution $\mathrm{P}(\mathbf{x}, y)$, which gives the probability for any combinations of observations and labels. The exact distribution is usually not known and the only clues we have about it are *samples* from the distribution (see figure 3.1). Therefore the challenge in learning lies in the *generalization*: to find an accurate label prediction function for data from the underlying distribution, only given a limited set of samples.

In general, the labels may be discrete or continuous values. However, in this work we only consider problems with discrete *class labels* from a finite domain. These are often referred to as *classification problems* (as opposed to *function approximation problems*). In the most basic case, the learner only has to distinguish two classes (*binary classification*,

cf. algorithm 4.1), and if there are more than two classes, the setup is called a *multiclass problem* (cf. algorithm 4.2). More generally, it is possible to consider problems where one has to predict a vector of labels $\mathbf{y}$ instead of a single label y, called a *relational learning problem*. These problems are harder to solve, because the learning algorithm not only needs to learn the dependency to the input $\mathbf{x}$ but also the dependencies between the components of the label vector $\mathbf{y}$ themselves (see also section 5.1 and algorithm 5.1). Relational learning is not to be confused with *multi-label problems*, where each input value may be labeled with a subset of a set of (scalar) labels $\{1, \ldots, J\}$ [cf. Schapire and Singer, 1999]. Here, the labels will always be mutually exclusive.

Mathematically, the problem can be defined as follows:

Definition 3.1. Let $P(\mathbf{x}, \mathbf{y})$ be the joint probability distribution of the (interdependent) random vectors $\mathbf{X}$ and $\mathbf{Y}$. Given a set of samples $\left\{(\mathbf{x}^{(s)}, \mathbf{y}^{(s)}) \;\middle|\; 1 \leq s \leq S\right\}$ from that distribution, the task in a supervised learning problem is to find a prediction function $H : \mathrm{dom}(\mathbf{X}) \mapsto \mathrm{dom}(\mathbf{Y})$ minimizing the generalization error

$$\mathrm{E}\big[f_{loss}(H(\mathbf{X}), \mathbf{Y})\big] \tag{3.1}$$

with respect to some loss function f_{loss}.

Remark 3.2. In classification problems, there is a straightforward loss function: either a label is correct or it isn't. It is possible to use other functions when some labels are considered more similar than others and hence confusing labels may imply variable loss values. However the *0/1 misclassification loss* is most common and shall also be used here. In relational learning, this loss function translates to the rate of incorrectly predicted labels in the label vector

$$f_{loss}(\mathbf{y}, \mathbf{y}') = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}_{\langle y_t = y'_t \rangle} \,. \tag{3.2}$$

So the aim of the learning algorithm is to minimize the overall rate of incorrect labels. Especially for the presentation of test results, it is also common to refer to the inverse measure $1 - \mathrm{E}\big[f_{loss}(H(\mathbf{X}), \mathbf{Y})\big]$, called the *correct classification rate* or simply *accuracy*.

Remark 3.3. The probability distribution $P(\mathbf{x}, \mathbf{y})$ is usually unknown. It is often governed by a complex, real-world process, which would be hard to capture exactly. However it is sufficient to have a set of samples from that distribution. In activity recognition tasks, these samples can be generated by naturally performing the activities we are interested in, recording the observations, and manually labeling them with the true labels. The *labeled data* is then partitioned into a *training set* and a *test set*, and only the training

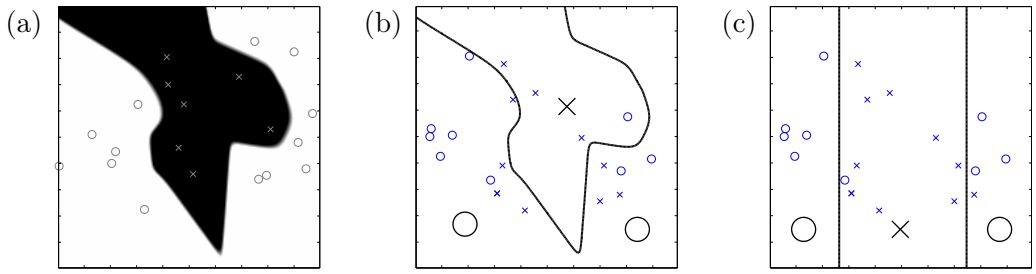

Figure 3.2: The samples from the actual distribution in figure 3.1c could also also have been the result of sampling from the non-smooth distribution in (a). The prediction function (b) which exactly matches the training set performs worse on new samples than the optimal predictor (c).

set is provided to the learning algorithm to find the prediction function $H(\mathbf{x})$. The quality of that function can then be assessed through the empirical expectation over the test set S_{test}

$$\mathrm{E}\big[f_{loss}(H(\mathbf{X}), \mathbf{Y})\big] \approx \frac{1}{|S_{test}|} \sum_{(\mathbf{x},\mathbf{y}) \in S_{test}} f_{loss}(H(\mathbf{x}), \mathbf{y}). \tag{3.3}$$

In order to reduce the effects caused by randomness in observations and labels, learning is repeated for different partitions and the resulting accuracy rates are averaged. In the common *leave-one-out* or *(n-fold) cross-validation* strategy, the data is split into n distinct sets, using any $n-1$ sets for training and the remaining set for testing.

Remark 3.4. Note that the probability distribution ranges over both the observations and the true labels. Hence the labels $\mathbf{Y}$ may not be determined given the observations $\mathbf{X}$ but there may be different labels possible for the same value of $\mathbf{X}$ (cf. the gray area in figure 3.1c). This is indeed the case in many real-world applications, where for example a certain sensor measurement may be caused by different activities. One consequence of this is that there is no "perfect" prediction function $H(\mathbf{x})$ which would yield an expected loss of zero.

More important is however that non-deterministic labels make the classification problem harder. This is because it is generally impossible to tell how smooth a distribution is, given only samples from that distribution. Many learning algorithms implicitly assume deterministic labels, i. e. conditional probabilities either zero or one like in figure 3.2a, and therefore are prone to *overfitting*. In that case, the prediction function accounts too much for the randomness in the training set, which decreases performance on the test set or any newly generated samples from the underlying distribution (see figure 3.2 b–c).

Chapter 4

LogitBoost

LogitBoost is a probabilistic learning algorithm for supervised learning by Friedman, Hastie, and Tibshirani [2000]. Its optimization strategy is the basis of *virtual evidence boosting* (VEB) and hence knowledge about LogitBoost is required to fully understand VEB. This chapter aims to provide a sound understanding of LogitBoost but also an introduction for readers who are not yet familiar with the method. We redo the formal derivation of LogitBoost and particularly focus on details which are hard to understand or even incomplete in the original paper [Friedman et al., 2000].

LogitBoost or other versions of boosting (cf. section 2.1) may be chosen as learning methods for an activity recognition task. Boosting is however limited in that it can always only predict scalar labels. In activity recognition we generally want to estimate temporal sequence of activity labels, so with boosting these would have to be estimated independently, i. e. by using only one input vector from a particular time slice to predict one label. Graphical models like *conditional random fields* (CRFs) on the other hand take all inputs and estimates the entire sequence at once. Therefore they can exploit dependencies between the labels like for example the fact that successive labels are often the same. We will revisit this issue in section 5.1 in the context of the introduction of VEB, a method for training these CRFs.

4.1 Fitting an Additive Model for Maximum Likelihood

In this section we will introduce the *model* maintained by LogitBoost and the optimization criterion for instantiating this model. The model can be thought of as the internal data structure to represent the prediction function $H(\mathbf{x})$ and in the case of probabilistic methods is usually a probability distribution or density function. LogitBoost models the

Input: the training data set $\big\{(\mathbf{x}^{(s)}, y^{(s)}) \;\big|\; 1 \le s \le S\big\}$; a function approximator for weighted least squares fitting; the number of boosting rounds M

Output: the classification function $H(\mathbf{x})$ predicting the labels $y \in \{0, 1\}$

initialize the model function $F \equiv 0$

for $m = 1$ **to** M **do**

  foreach sample index $s \in \{1, \ldots, S\}$ **do**

    for the input vector $\mathbf{x}^{(s)}$ compute the modeled probability for $y = 1$, the so-called *working response*, and the weight:

$$p^{(s)} = \frac{e^{F(\mathbf{x}^{(s)})}}{e^{F(\mathbf{x}^{(s)})} + e^{-F(\mathbf{x}^{(s)})}}$$
$$z^{(s)} = \frac{y^{(s)} - p(\mathbf{x}^{(s)})}{p(\mathbf{x}^{(s)})(1 - p(\mathbf{x}^{(s)}))}$$
$$w^{(s)} = p(\mathbf{x}^{(s)})(1 - p(\mathbf{x}^{(s)}))$$

  end

  use the function approximator to fit the function $f_m(\mathbf{x})$ to $(\mathbf{x}^{(s)}, z^{(s)})$ by weighted least squares with weights $w^{(s)}$

  update the model $F \leftarrow F + \frac{1}{2} f_m$

end

output the classifier

$$H(\mathbf{x}) = \begin{cases} 1 & \text{if } F(\mathbf{x}) > 0 \quad (\text{or equivalently } \sum_{m=1}^{m} f_m(\mathbf{x}) > 0) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 4.1: LogitBoost (2 classes)

conditional probability for the category labels given the input $\mathrm{P}(Y = j \mid \mathbf{X} = \mathbf{x})$. In this section and the following we will present LogitBoost for binary classification problems (algorithm 4.1), i. e. with only two possible classes $j = 0$ and $j = 1$, before generalizing the algorithm for multiclass problems in section 4.4.

The input for the LogitBoost algorithm is a set of pairs of inputs $\mathbf{x}$ and the corresponding true label y. It is assumed that these pairs are instantiations of the random variables $\mathbf{X}$ and Y (cf. section 3.2) and that there exists some (usually unknown) conditional probability distribution $\mathrm{P}(Y \mid \mathbf{X})$. LogitBoost aims to approximate this distribution because if it was known the optimal prediction function would be straightforward:

$$H(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathrm{P}(Y = 1 \mid \mathbf{x}) > 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Without loss of generalization only the probability $\mathrm{P}(Y = 1 \mid \mathbf{x})$ needs to be represented, which is a function of $\mathbf{x}$ with values in $[0, 1]$. LogitBoost finds this function, which we call the *model function*, through an optimization process. This is an unusual application of optimization because the optimization variable is a function instead of just a scalar or vector. Since probabilities are restricted to values in $[0, 1]$, the model function should also meet this restriction. This would however make the optimization much harder since it introduces a constraint to the otherwise unconstrained optimization. To avoid this difficulty, the conditional probabilities are mapped to $\mathbb{R}$ and result is represented by the model function F instead. A convenient map is the monotone *logit transformation* which yields the following target:

$$F(\mathbf{x}) \approx \frac{1}{2} \log \frac{\mathrm{P}(Y = 1 \mid \mathbf{x})}{\mathrm{P}(Y = 0 \mid \mathbf{x})}. \tag{4.2}$$

Given the model function, the probabilities can be easily recovered:

$$\mathrm{P}(Y = 1 \mid \mathbf{x}) \approx \frac{e^{F(\mathbf{x})}}{e^{F(\mathbf{x})} + e^{-F(\mathbf{x})}} \quad \stackrel{\text{def}}{=} \quad \mathrm{P}(Y = 1 \mid \mathbf{x}; F). \tag{4.3}$$

The conditional probability with the model function F as extra argument on the conditional side stands for the conditional probability which is *modeled* — or equivalently: represented — by the algorithm.[1] The prediction function is based on the modeled probability and set to 1 if and only if $\mathrm{P}(Y = 1 \mid \mathbf{x}; F) > 0.5$, or equivalently $F(\mathbf{x}) > 0$

With $\mathbf{x}$ being a vector with real-valued entries, it is not possible to computationally

[1] In the following the model function may be omitted for brevity because we will — unless explicitly stated otherwise — always be talking about modeled probabilities.

represent all functions $F : \mathrm{dom}(\mathbf{x}) \mapsto \mathbb{R}$. It is necessary to choose a family of functions which can be parameterized with a finite number of parameters. In LogitBoost F is allowed to be a finite sum of some simple *base functions* f_m which each only require a few parameters. With equation (4.3) the parameterization of F implies a family of probability distributions that can be represented by LogitBoost. In the probabilistic learning literature, the term *model* is used to denote this family of distributions (as in "graphical model") but also sometimes stands for the single distribution instance found by a learning method (as in "to fit a model"). We attempt to make it clear if we refer to instance by using the expression "model function", but some ambiguity can't be avoided. Friedman et al. [2000] for example coined the expression "LogitBoost fits an additive logistic model," which summarizes that the method finds a model function taking the form $F(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$ to approximate the logistic transformation of the probabilities.

The model function is determined in a forward stepwise manner. In each iteration of the algorithm, a new function f_m is chosen from a set of base function and added to the model to improve the optimization criterion (see following paragraph). Each of the updates is done in a greedy fashion: once added a function will never be revised or removed. The base functions are all those functions that can be represented by a particular *function approximator*. Popular choices include *decision trees*, *neural nets*, and *kernel functions* [Meir and Rätsch, 2003, chapter 8], and even with the most basic function approximator, a *step function*

$$f_{step}(\mathbf{x}) = \begin{cases} a & \text{if } x_c < t, \\ b & \text{if } x_c \geq t, \end{cases} \tag{4.4}$$

very good results can be achieved [Friedman et al., 2000, chapter 7]. Step functions are commonly referred to as *decision stumps* because they are equivalent to decision trees with only a single level. Note that it is assumed that for any base function f the function approximator can equally represent scaled versions $\alpha \cdot f$ of that function. Therefore the restriction of F to be a simple sum instead of a weighted sum does not affect the model, i. e. the family of distributions that can be represented.

The criterion maximized by LogitBoost is the *log-likelihood*. (For the distinction of other criteria that contain the word "likelihood", the term as defined here shall also be referred to as *data likelihood*.) It is defined as the natural logarithm of the joint conditional probability of the true labels $y^{(s)}$ of all samples in the training set given all input values $\mathbf{x}^{(s)}$:

$$\log \mathrm{P}(y^{(1)} \ldots, y^{(S)} \,|\, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(S)}; F). \tag{4.5}$$

This conditional probability depends on the current model function F. In fact, with the training set being fixed, this value only depends on the model function and shall be written in short as $\ell(F)$. Since we assume that sample pairs $(\mathbf{x}^{(s)}, y^{(s)})$ are independently generated (cf. definition 3.1), the log-likelihood computes as

$$\ell(F) \stackrel{\text{def}}{=} \sum_{s=1}^{S} \log \mathrm{P}(Y = y^{(s)} \,|\, \mathbf{x}^{(s)}; F) \tag{4.6}$$

The logarithm in the optimization criterion is taken to simplify the computations. This is permissible because log is a strictly monotone function, and hence the (local and global) maxima of the likelihood and the log-likelihood coincide.

While this optimization criterion can't guarantee that the modeled conditional probability will accurately match the unknown true probability if the number of samples finite (cf. figure 3.1), we at least know that a maximum likelihood estimator makes best use of the information it is given if the number of samples reaches infinity (it is *asymptotically efficient*) [Kay, 1993]. In this case the likelihood is maximized if the model function is equals the true conditional probability [Friedman et al., 2000], i. e.

$$\mathrm{P}(Y = 1 \,|\, \mathbf{x}; F) = \mathrm{P}(Y = 1 \,|\, \mathbf{x}) \tag{4.7}$$

for all $\mathbf{x}$. We will discuss practical implications of the criterion is section 4.3.

4.2 Optimizing the Likelihood with Newton's Method

In this section we will show that LogitBoost maximizes the data likelihood by an optimal approximations to Newton steps. *Newton's method for optimization* is a well-known and (under certain conditions) quickly converging gradient descent method. Here however we are optimizing over functions $f : \mathrm{dom}(\mathbf{X}) \mapsto \mathbb{R}$ which causes the difficulty that the Newton update is generally impossible to represent. Recall that in each iteration, LogitBoost adds a single base function f and the difficult question is how to choose f if none of the available functions exactly matches the Newton step. Friedman et al. [2000] derive a weighted least square approximation, however the optimality of that approximation remains vague.

This issue can overcome by basing our derivation on a different view on Newton's method: The update steps are equivalent to analytically finding the optimum with respect to the 2^{nd} order Taylor polynomial of the target function. This means that even if the optimum can't be resented, it is now possible to tell which base function would yield the best update. It can be shown that each LogitBoost update maximizes the Taylor

approximation of the likelihood.

Theorem 4.1. *In each iteration LogitBoost augments its additive logistic model by the base function that maximizes the 2^{nd} order Taylor polynomial of the data likelihood.*

Before proving this we introduce the following notation:

Definition 4.2. Let $\{(\mathbf{x}^{(s)}, y^{(s)}) \mid 1 \le s \le S\}$ be a set of instantiations of the random variables $(\mathbf{X}, Y)$ (cf. definition 3.1). Then the *empirical expectation* $\hat{\mathrm{E}}$ over $\mathbf{X}$ and Y is defined as

$$\hat{\mathrm{E}}\big[g(\mathbf{X}, Y)\big] = \frac{1}{S}\sum_{s=1}^{S} g(\mathbf{x}^{(s)}, y^{(s)}).$$

Given the weights $w : \mathrm{dom}(\mathbf{X}) \times \mathrm{dom}(Y) \mapsto \mathbb{R}$, the *weighted empirical expectation* $\hat{\mathrm{E}}_w$ is defined as

$$\hat{\mathrm{E}}_w\big[g(\mathbf{X}, Y)\big] = \hat{\mathrm{E}}\big[w(\mathbf{X}, Y)\, g(\mathbf{X}, Y)\big] = \frac{1}{S}\sum_{s=1}^{S} w(\mathbf{x}^{(s)}, y^{(s)})\, g(\mathbf{x}^{(s)}, y^{(s)})$$

Proof of theorem 4.1. LogitBoost represents the probability $\mathrm{P}(Y = 1 \mid \mathbf{X} = \mathbf{x}; F)$ by

$$p(\mathbf{x}) \stackrel{\mathrm{def}}{=} \frac{e^{F(\mathbf{x})}}{e^{F(\mathbf{x})} + e^{-F(\mathbf{x})}}. \tag{4.8}$$

Since the random variable Y can only take values $y \in \{0, 1\}$, the conditional probability $\mathrm{P}(Y = y \mid \mathbf{X} = \mathbf{x}; F)$ can be written as $y\,p(\mathbf{x}) + (1-y)(1-p(\mathbf{x}))$. This yields the following form for the data likelihood (4.6):

$$\begin{aligned}
\ell(F) &= S \cdot \hat{\mathrm{E}}\big[\log \mathrm{P}(Y \mid \mathbf{X}, F)\big] \\
&= S \cdot \hat{\mathrm{E}}\big[Y \log\big(p(\mathbf{X})\big) + (1 - Y) \log\big(1 - p(\mathbf{X})\big)\big] \\
&= S \cdot \hat{\mathrm{E}}\big[Y F(\mathbf{X}) + (1 - Y)(-F(\mathbf{X})) - \log\big(e^{F(\mathbf{X})} + e^{-F(\mathbf{X})}\big)\big] \\
&= S \cdot \hat{\mathrm{E}}\big[2Y F(\mathbf{X}) - \log\big(1 + e^{2F(\mathbf{X})}\big)\big]
\end{aligned} \tag{4.9}$$

Since the sample set size S is constant, it is irrelevant for maximizing the data likelihood and will be dropped in the following equations.

In the update step a base function f is added to the current model F, resulting in the data likelihood $\ell(F + f)$. Given the derivatives

$$\begin{aligned}
\frac{\partial \log \mathrm{P}(y \mid \mathbf{x}, F + f)}{\partial f(\mathbf{x})}\Big|_{f \equiv 0} &= 2y - 2\frac{e^{2(F(\mathbf{x}) + f(\mathbf{x}))}}{1 + e^{2(F(\mathbf{x}) + f(\mathbf{x}))}}\Big|_{f \equiv 0} = 2(y - p(\mathbf{x})) \\
\frac{\partial^2 \log \mathrm{P}(y \mid \mathbf{x}, F + f)}{\partial^2 f(\mathbf{x})}\Big|_{f \equiv 0} &= -4\frac{e^{2(F(\mathbf{x}) + f(\mathbf{x}))}}{(1 + e^{2(F(\mathbf{x}) + f(\mathbf{x}))})^2}\Big|_{f \equiv 0} = -4p(\mathbf{x})(1 - p(\mathbf{x})),
\end{aligned}$$

the new likelihood $\ell(F+f)$ can be approximated by Taylor expansion around the current working point $F+0$ or $f \equiv 0$:

$$\begin{aligned}\ell(F+f) &\approx \hat{\mathrm{E}}\big[\log \mathrm{P}(Y \mid \mathbf{X}, F) + f(\mathbf{X}) \cdot 2\,(Y - p(\mathbf{X})) - f(\mathbf{X})^2 \cdot 2\,p(\mathbf{X})(1-p(\mathbf{X}))\big] \\ &= \hat{\mathrm{E}}\big[\log \mathrm{P}(Y \mid \mathbf{X}, F)\big] - 2\,\hat{\mathrm{E}}_w\big[f(\mathbf{X})^2 - \frac{Y - p(\mathbf{X})}{p(\mathbf{X})(1-p(\mathbf{X}))} f(\mathbf{X})\big],\end{aligned}$$

with weights $w(\mathbf{x}, y) = p(\mathbf{x})(1-p(\mathbf{x}))$.

The aim is now to find the base function f that maximizes this expression. It can be easily verified that the approximate data likelihood reaches its maximum if and only if the following is minimized:[2]

$$\begin{aligned}&\hat{\mathrm{E}}_w\big[f(\mathbf{X})^2 - \tfrac{Y - p(\mathbf{X})}{p(\mathbf{X})(1-p(\mathbf{X}))} f(\mathbf{X}) + \tfrac{1}{4}\big(\tfrac{Y - p(\mathbf{X})}{p(\mathbf{X})(1-p(\mathbf{X}))}\big)^2\big] \\ &= \hat{\mathrm{E}}_w\big[\big(f(\mathbf{X}) - \tfrac{1}{2}\tfrac{Y - p(\mathbf{X})}{p(\mathbf{X})(1-p(\mathbf{X}))}\big)^2\big].\end{aligned} \tag{4.10}$$

The function approximator called in every iteration of LogitBoost fits the base function $f(\mathbf{x})$ to $\big(\mathbf{x}^{(s)}, \frac{1}{2}\frac{y^{(s)} - p(\mathbf{x}^{(s)})}{p(\mathbf{x}^{(s)})(1-p(\mathbf{x}^{(s)}))}\big)$ by *weighted least-squares* using the weights as defined above. Therefore f minimizes expression (4.10) and the update has the claimed optimality. □

Note that in algorithm 4.1 the factor $\frac{1}{2}$ is multiplied after fitting f, which yields the same updates as in the proof. This is done to emphasize the similarity to the multiclass version of LogitBoost (algorithm 4.2).

4.3 Discussion and Evaluation

From a theoretic point of view, LogitBoost has very promising properties. It applies a fast converging optimization method — a best effort approximation of the Newton method — to the statistically sound optimization criterion maximum likelihood. Also the chosen parameterization, an additive logistic regression model, is popular in the statistics community [Friedman et al., 2000, Hastie and Tibshirani, 1990]. This is supported by very good experimental results on real-world and simulated data, equivalent or surpassing that of AdaBoost [Friedman et al., 2000].

However, for a theoretic evaluation one also needs to consider the following two issues. LogitBoost may be unstable like any method based on Newton updates. In that case *GentleBoost* also proposed by Friedman et al. [2000] may be more appropriate. It

[2] We can add and remove terms that don't depend on f. For example $p(\mathbf{x})$ only depends on the current model function $F(\mathbf{x})$ but not the additive update $f(\mathbf{x})$.

optimizes an approximation of the data likelihood and so yields bounded updates.

Secondly, LogitBoost doesn't perform any explicit *regularization*, which would mean adding a penalizing term for unlikely models to the optimization criterion. Unless the sample set contains samples with matching input $\mathbf{x}^{(s)}$ and different labels $y^{(s)}$, the highest data likelihood could be achieved by setting the modeled conditional probability $\mathrm{P}(Y \mid \mathbf{X}; F)$ to either 0 or 1 to exactly match every sample (cf. figure 3.2a). Usually we know that these non-smooth distributions are less likely than smoother ones, but LogitBoost (like most other boosting versions) would still aim for the non-smooth distribution. Still, it depends on the base functions if complex distributions can be represented, so some regularization is also achieved with an appropriate choice base functions (e. g. not too expressive ones). Nevertheless, LogitBoost may overfit, although it was like AdaBoost thought to be resistant to overfitting [Mease and Wyner, 2005]. This issue especially occurs if the data is *noisy*, i. e. if there are ranges of $\mathbf{X}$ where the true conditional probability $\mathrm{P}(Y = y \mid \mathbf{X})$ is neither close to zero or one (cf. figure 3.1c). This problem can be explicitly addressed by a different choice of optimization criterion [Mason et al., 2000, Meir and Rätsch, 2003].

4.4 LogitBoost for Multiclass Problems

In this section it is shown how LogitBoost can be generalized to multiclass problems. The task is hence to find a prediction function which can predict the correct label from the set $\{1, \ldots, J\}$ instead of just binary labels. The idea is again to model the probabilities $\mathrm{P}(Y = j \mid \mathbf{x})$ for each class and then simply predict the most likely label for the given input $\mathbf{x}$. The key difference is that we need more than one model function to represent all class probabilities. Since the probabilities have to sum to one, $J - 1$ model functions would be sufficient, but for reasons which will be discussed in remark 4.6, each class j gets its own model function F_j. The algorithm we are describing in this section is listed as algorithm 4.2.

For the same reasons as before, i. e. to simplify the optimization process, the model function should be allowed to take any real values. Therefore, each class probability is mapped to $\mathbb{R}$ and that value is represented by $F_j(\mathbf{x})$. The mapping used is a symmetric generalization of the two-class logistic transformation. It is defined as follows:

Input: the training data set $\big\{(\mathbf{x}^{(s)}, y^{(s)}) \;\big|\; 1 \le s \le S\big\}$; a function approximator for weighted least squares fitting; the number of boosting rounds M

Output: the classification function $H(\mathbf{x})$ predicting the labels $y \in \{0, \ldots, J\}$

initialize the model functions $F_j \equiv 0$
for $m = 1$ **to** M **do**
  foreach class label $j \in \{1, \ldots, J\}$ **do**
    foreach sample index $s \in \{1, \ldots, S\}$ **do**
      for the input vector $\mathbf{x}^{(s)}$, compute the modeled probability for $y = j$, the respective working response and weight:

$$p_j(\mathbf{x}^{(s)}) = \frac{e^{F_j(\mathbf{x}^{(s)})}}{\sum_{k=1}^{J} e^{F_k(\mathbf{x}^{(s)})}}$$
$$z_j^{(s)} = \frac{\mathbf{1}_{\langle y^{(s)}=j \rangle} - p_j(\mathbf{x}^{(s)})}{p_j(\mathbf{x}^{(s)})(1 - p_j(\mathbf{x}^{(s)}))}$$
$$w_j^{(s)} = p_j(\mathbf{x}^{(s)})(1 - p_j(\mathbf{x}^{(s)}))$$

    end
    use the function approximator to fit the function $f_{m,j}(\mathbf{x})$ to $(\mathbf{x}^{(s)}, z_j^{(s)})$ by weighted least squares with weights $w_j^{(s)}$
  end
  foreach class label $j \in \{1, \ldots, J\}$ **do**
    compute normalized updates $f'_{m,j}(\mathbf{x}) = \frac{J-1}{J}\big(f_{m,j}(\mathbf{x}) - \frac{1}{J}\sum_{k=1}^{J} f_{m,k}(\mathbf{x})\big)$
    update the model function $F_j \leftarrow F_j + f'_{m,j}$
  end
end
output the prediction function $H(\mathbf{x}) = \arg\max_j F_j(\mathbf{x})$

Algorithm 4.2: LogitBoost (J classes)

Definition 4.3. Let $p_j(\mathbf{x})$ abbreviate the probabilities $\mathrm{P}(Y = j \mid \mathbf{x})$. Then the *symmetric multiple logistic transformation* is defined as

$$F_j(\mathbf{x}) = \log \frac{p_j(\mathbf{x})}{\sqrt[J]{\prod_{k=1}^{J} p_k(\mathbf{x})}} \tag{4.11a}$$

$$= \log p_j(\mathbf{x}) - \frac{1}{J} \sum_{k=1}^{J} \log p_k(\mathbf{x}). \tag{4.11b}$$

This is equivalent to

$$p_j(\mathbf{x}) = \frac{e^{F_j(\mathbf{x})}}{\sum_{k=1}^{J} e^{F_k(\mathbf{x})}}, \qquad \sum_{k=1}^{J} F_k(\mathbf{x}) = 0. \tag{4.12}$$

Remark 4.4. The equivalence of the two definitions can be easily shown by substituting all occurrences of F in (4.12) by equation (4.11a) and by substituting p in (4.11b) by the definition in (4.12). The sum constraint in (4.12) is needed to ensure that the parameterization is unique because the probabilities $p_j(\mathbf{x})$ would not change if a constant was added to all $F_k(\mathbf{x})$. In theory the sum may be fixed to any constant value, but the choice of zero is entailed by the definition (4.12) which always produces a zero-centered set of model functions.

The alternative to the parameterization defined above is the *standard multiple logistic transformation*, which applies the idea of only using $J - 1$ model functions. Since that parameterization is required as intermediate representation for the proof of multiclass LogitBoost, it is also introduced here.

Definition 4.5. In the standard multiple logistic transformation, the label probabilities $p_j(\mathbf{x})$ are represented by the model functions

$$G_j(\mathbf{x}) = \log \frac{p_j(\mathbf{x})}{p_1(\mathbf{x})} = \log p_j(\mathbf{x}) - \log p_1(\mathbf{x}). \tag{4.13}$$

An equivalent definition is

$$p_j(\mathbf{x}) = \frac{e^{G_j(\mathbf{x})}}{\sum_{k=1}^{J} e^{G_k(\mathbf{x})}}, \qquad G_1(x) = 0. \tag{4.14}$$

Remark 4.6. The *base class*, here class 1, can be arbitrarily chosen. This is the disadvantage of this parameterization because it is often observed that the choice of base class affects the classification performance. Since the probability $p_1(\mathbf{x})$ does not have it's own (nontrivial) model function component $G_1(\mathbf{x})$, the optimization of the learning algorithm

can't focus on getting $p_1(\mathbf{x})$ right but that probability is only affected by the updates to the other components. Unless there is a special default class like 'no label', this effect is not desired.

Descriptively, the symmetric and standard multiple logistic transformations differ in what each of the model components F_j or G_j represents. For the standard transformation they can be interpreted as the confidence for label j versus the base class (*one-vs-reference* parameterization). The symmetric transformation on the other hand represents the confidence for a class versus all other classes (*one-vs-all* parameterization).

As before, each of the model functions F_j is additive, i. e. it is built from sequentially adding base functions $f_{m,j}(\mathbf{x})$. This makes the optimization of the likelihood a bit tricky since the symmetric parameterization imposes the centering constraint (4.12) on the model functions. One way to deal with this would be to ignore the constraint during the update and just re-center the models afterwards. However it turns out that this way the computed steps are too large by $\frac{J}{J-1}$, and hence if that approach was applied to the two-class case, the effective update would be twice the optimal update.

So the solution chosen by Friedman et al. [2000] is to shift the parameterization to the standard multiple logistic transformation, update those model functions, and convert the result back to the symmetric parameterization. In order to eliminate effects of the choice of base class (cf. remark 4.6), the updates are computed with each possible base class and the average is applied. Effectively, this only yields the factor $\frac{J-1}{J}$ and hence means no extra computational cost. Still, this is a key idea for the derivation of the multiclass version of LogitBoost.

The data likelihood is again optimized by Newton steps. However instead of a scalar function $F(\mathbf{x})$, the optimization variable is the vector of functions $(F_j(\mathbf{x}))_{j=2\ldots J}$. Since the inversion of the Hessian matrix, a $(J-1)$-dimensional matrix of functions of $\mathbf{x}$, is generally not possible, full Newton stepping is not an option. Instead, the 2^{nd} derivative is approximated by a diagonal matrix, resulting in a quasi-Newton update. The following theorem summarizes the characteristics and optimality of the update step.

Theorem 4.7. *In each iteration LogitBoost (J classes) augments its model by an average of quasi-Newton steps maximizing the data likelihood. The steps are computed w. r. t. the standard multiple logistic transformation of the label probabilities for each possible base class. Where the update can't be represented by the base functions, the best approximation in terms of quasi-Newton steps is chosen.*

Proof. With the definitions above (equations 4.11 and 4.13), it is evident that

$$G_j(\mathbf{x}) = F_j(\mathbf{x}) - F_1(\mathbf{x}) \tag{4.15}$$

defines the standard multiple logistic model of the probabilities $p_j(\mathbf{x})$ with base class 1. Rewriting the modeled probabilities $\mathrm{P}(Y \mid \mathbf{X}, G)$, the data likelihood is given by

$$\begin{aligned}\ell(G) &= \hat{\mathrm{E}}\big[\log \mathrm{P}(Y \mid \mathbf{X}, G)\big] \\ &= \hat{\mathrm{E}}\Big[\sum_{j=1}^{J} \mathbf{1}_{\langle Y=j\rangle} G_j(\mathbf{X}) - \log \sum_{j=1}^{J} e^{G_j(\mathbf{X})}\Big]\end{aligned} \tag{4.16}$$

A generic update adds the functions $g_2, \ldots, g_J$ to the respective model functions resulting in the new model $G + g$. Note that in compliance with (4.14) there there is no update for the base class model function G_1, i. e. $g_1 \equiv 0$.

Similar to the two-class case, these functions are determined by finding the maximum of the 2nd order Taylor approximation of the data likelihood around the current model $G + 0$. Since the update is *quasi-Newton*, i. e. the Hessian matrix is approximated, the same approximation is used in the Taylor polynomial. Here that amounts to Hessian matrix with all off-diagonal elements set to zero. With the derivatives

$$\frac{\partial \log \mathrm{P}(y \mid \mathbf{x}, G + g)}{\partial g_j(\mathbf{x})}\Big|_{g \equiv 0} = \mathbf{1}_{\langle y=j\rangle} - p_j(\mathbf{x}) \tag{4.17}$$

$$\frac{\partial^2 \log \mathrm{P}(y \mid \mathbf{x}, G + g)}{\partial^2 g_j(\mathbf{x})}\Big|_{g \equiv 0} = -p_j(\mathbf{x})(1 - p_j(\mathbf{x})), \tag{4.18}$$

the data likelihood (cf. 4.16) is approximately

$$\begin{aligned}\ell(G + g) &\approx c + \hat{\mathrm{E}}\Big[\sum_{j=2}^{J} g_j(\mathbf{X}) \cdot (\mathbf{1}_{\langle y=j\rangle} - p_j(\mathbf{X})) - \frac{1}{2}\sum_{j=2}^{J} g_j(\mathbf{X})^2 \cdot p_j(\mathbf{X})(1 - p_j(\mathbf{X}))\Big] \\ &= c - \frac{1}{2}\sum_{j=2}^{J} \hat{\mathrm{E}}_{w_j}\Big[g_j(\mathbf{X})^2 - 2\,\frac{\mathbf{1}_{\langle Y=j\rangle} - p_j(\mathbf{X})}{p_j(\mathbf{X})(1 - p_j(\mathbf{X}))}\, g_j(\mathbf{X})\Big],\end{aligned} \tag{4.19}$$

with weights $w_j(\mathbf{x}, y) = p_j(\mathbf{x})(1 - p_j(\mathbf{x}))$ and $c = \hat{\mathrm{E}}\big[\log \mathrm{P}(Y \mid \mathbf{X}, G)\big]$, a constant term with respect to the update g. Now it is easy to verify (cf. proof of theorem 4.1) that the likelihood is maximized if the update functions g_j $(j = 2, \ldots, J)$ are fitted to $\big(\mathbf{X}, \frac{\mathbf{1}_{\langle Y=j\rangle} - p_j(\mathbf{X})}{p_j(\mathbf{X})(1-p_j(\mathbf{X}))}\big)$ by weighted least square. Note that the fitted functions g_j don't depend on the choice of base class.

The parameterization of the model after the update $G + g$ can be converted back to the symmetric multiple logistic transformation by adding $-\frac{1}{J}\sum_{j=1}^{J} G_j + g_j$ to each of the model components. (It is easy to see that the parameterizations before and after this step represent the same probabilities $p_j(\mathbf{x})$; see (4.12) and (4.14).) Using the definition from the forward conversion (4.15) this results in the following symmetric model after

the update:

$$\begin{aligned} F_j + f_j &\equiv G_j + g_j - \frac{1}{J}\sum_{k=1}^{J} G_k + g_k \\ &\equiv F_j + \big(g_j - \frac{1}{J}\sum_{k=1}^{J} g_k\big) \end{aligned} \tag{4.20}$$

This procedure can be repeated for every choice of base class b. It yields the updates $f_j \equiv 0$ if $j = b$ and $f_j \equiv g_j - \frac{1}{J}\sum_{j=1}^{J} g_j$ otherwise. Averaging over all base classes hence gives the update

$$f'_j(\mathbf{x}) = \frac{J-1}{J}\big(g_j(\mathbf{x}) - \frac{1}{J}\sum_{k=1}^{J} g_k(\mathbf{x})\big). \tag{4.21}$$

This concludes the derivation of multiclass version of LogitBoost (algorithm 4.2). □

Chapter 5

Virtual Evidence Boosting

Virtual evidence boosting (VEB) is a efficient method for training *conditional random fields* (CRFs) of arbitrary topology with discrete and continuous features. The reader who is familiar with CRFs and the inference method *belief propagation* method may directly skip to section 5.4 where the novel method virtual evidence boosting is presented. For all others the following sections introduce the remaining foundations of VEB.

5.1 Motivation for Graphical Models

Statistical learning methods seek a probability distribution over the input $\mathbf{X}$ and the labels $\mathbf{Y}$ from a predefined family of distributions, the so-called *model*, to underpin the required prediction function $H(\mathbf{x})$. The key limitation of LogitBoost is that its model only allows for scalar labels Y and hence only independent labels can be predicted. The natural way of parameterizing distributions over multiple dependent random variables is through graphical models. An example of such a model formalism are conditional random fields, which are introduced in the following section. Other well-known graphical models are *Hidden Markov Models* and *Dynamic Bayesian networks* (cf. section 2.2). Generally these models have the common idea to represent the random variables as network nodes and to indicate dependencies by edges or arcs connecting the dependent variables. A special type of graphical models are the so-called *temporal models* where the nodes represent a state variable at different points in time.

Applied to our problem of recognizing spatial context, the graph for the temporal model would look as depicted in figure 5.1. From the data streams produced by the wearable sensor, *features* are computed for a series of times $(1, \ldots, T)$. Although the choice of features is important for the application (see section 6.1), for now it is sufficient

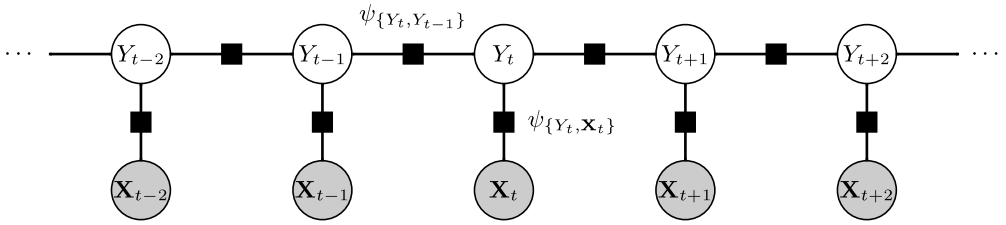

Figure 5.1: A linear chain conditional random field.

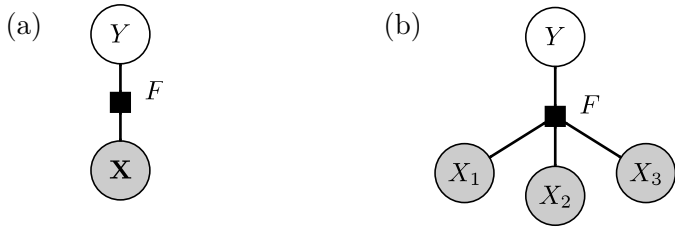

Figure 5.2: The model of LogitBoost written as conditional random field. The observed node labeled with the random vector $\mathbf{X}$ in (a) is an abbreviation for several nodes. For $\mathbf{X} = (X_1\, X_2\, X_3)^T$ figure (a) is equivalent to figure (b).

to know that they yield a vector $\mathbf{x}_t$, the so-called *feature values* or *observations*, for each $t \in \{1, \ldots, T\}$. Furthermore there is a correct spatial context label y_t for each point in time. In order to model this data statistically, the values $\mathbf{x}_t$ and y_t are considered to be instantiations of the random variables $\mathbf{X}_t$ and Y_t. Since the spatial context of a person only changes occasionally, there is a strong correlation between successive label variables Y_t and Y_{t+1}. Also, by design of the features there are correlations between the label variables Y_t and the respective features $\mathbf{X}_t$. Once the model parameters are learned, it is possible to infer a consistent set of labels, taking the dependencies between the labels into account.

A thinkable alternative to fitting a graphical model is to treat the feature/label pairs as independent samples $\big\{(\mathbf{x}_t, y_t) \,|\, t = 1, \ldots, T\big\}$ of the random variables $\mathbf{X}$ and Y and then to apply LogitBoost (algorithm 4.2). Although this may indeed work, it cannot exploit the correlations between successive labels and therefore generally yields inferior results (see section 6.3.2). The model of LogitBoost can also be seen as a simple graphical model (figure 5.2).

Remark 5.1. From a statistical point of view it is not absolutely necessary to use a temporal model for activity recognition. This is the case if $\mathbf{X}_t$ is a sufficient statistic for Y_t, and hence Y_t and $Y_{t'}$ are conditionally independent given their feature values $\mathbf{X}_t$

and $\mathbf{X}_{t'}$. This can approximately be achieved by using features that are computed over highly overlapping ranges of the sensor data streams. The activity recognition application presented by Bao and Intille [2004] implements this idea, however without referring to the argument given here as to why this may be statistically sound. Nevertheless, explicitly modeling the relation between labels clearly surpasses that approach in the scalability to strong correlations and complex models.

5.2 The Statistical Model: Conditional Random Fields

Conditional random fields are graphical models for conditional distributions [Lafferty et al., 2001] and are therefore particularly suitable for activity recognition tasks (cf. section 2.2). They model the joint conditional probability of the label variables $\mathbf{Y}$ given the input variables $\mathbf{X}$. (Other names for the label variables are *hidden variables* because they can't be observed in the application, or *output variables*). A very good tutorial which presents conditional random fields in comparison to other graphical model has been published by Sutton and McCallum [2006]. Nevertheless the definition — including the generalizations permitted if virtual evidence boosting is used for training the parameters — is repeated here.

A conditional random field defines a family of conditional distributions $\mathrm{P}(\mathbf{Y} \mid \mathbf{X})$ that factorize according to a certain graph. This graph (cf. figures 5.1 and 5.2), called a *factor graph* [Kschischang et al., 2001], is bipartite with two types of nodes: *variable nodes* (depicted as circles) representing the random variables and *factor nodes* (depicted as black squares). The random variables can again be distinguished into two groups: the input variables X_t which are known during both training and testing, and the label variables Y_t which are to be predicted. It is required that the labels Y_t can only take discrete values from a finite set, written as $\mathrm{dom}(Y_t)$. The input variables however can take any value from a continuous domain. This generalization, possible due to VEB training, is very important for activity recognition tasks because most low-level features extracted from the sensors yield rational numbers (cf. section 6.1). Finally, the nodes corresponding to label variables are referred to as *label nodes* or *hidden nodes*, as opposed to the *observed nodes* of input variables.

Each of the factor nodes (or just *factors*) $\psi_A \in \Psi$ connects a certain number of variable nodes, which is specified by the set $A \subset \{\mathbf{X}_t \mid t = 1, \ldots, T\} \cup \{Y_t \mid t = 1, \ldots, T\}$. The factors ψ_A also stand for functions $\psi_A : \mathrm{dom}(\mathbf{X}_{\langle A \rangle}) \times \mathrm{dom}(\mathbf{Y}_{\langle A \rangle}) \mapsto \mathbb{R}^+$ over their neighbors, representing the compatibility of assignments to these variables. (Other names for factor functions are *compatibility functions* or *local functions*). Together they define

the probability distribution represented by the conditional random field. The probability of a labeling $\mathbf{y}$ given the input variable values $\mathbf{x}$ computes as the product of the local factors:

$$\mathrm{P}(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\psi_A \in \Psi} \psi_A(\mathbf{x}_{\langle A \rangle}, \mathbf{y}_{\langle A \rangle}), \tag{5.1}$$

with the normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{\psi_A \in \Psi} \psi_A(\mathbf{x}_{\langle A \rangle}, \mathbf{y}_{\langle A \rangle}). \tag{5.2}$$

The factor functions are assumed to be of the form

$$\psi_A(\mathbf{x}_{\langle A \rangle}, \mathbf{y}_{\langle A \rangle}) = \exp\{F_A(\mathbf{x}_{\langle A \rangle}, \mathbf{y}_{\langle A \rangle})\} = \exp\Big\{\sum_m f_{A,m}(\mathbf{x}_{\langle A \rangle}, \mathbf{y}_{\langle A \rangle})\Big\} \tag{5.3}$$

for arbitrary choices of $f_{A,m}$ from a infinite set of base functions (see section 5.4.1 for details). Note that the permissible *model functions* F_A are more general than those in the common conditional random fields [Lafferty et al., 2001, Sutton and McCallum, 2006], where F_A has to be a weighted sum over a fixed set of base functions.[1] In that context only the weights are learned, where here the full set of function parameters, for example the threshold of a step function, are chosen during training (cf. the LogitBoost model, section 4.1). Again, the sum in (5.3) does not need to be weighted because the set of base functions is assumed to be closed under multiplication with scalar values and hence the weight would be redundant.

The factors ψ_A of a conditional random field determine two important properties of the model: which pairs or cliques of random variables are dependent and how these dependencies are parameterized. The network topology is usually implied by the semantics of the random variables in the specific application. A very common case are *linear chain CRFs*, which are similar to hidden Markov models. Each label variable Y_t, $t \in \{1, \ldots, T\}$ depends on its predecessor and successor labels Y_{t-1} and Y_{t+1}, as well as its feature vector $\mathbf{X}_t$. This can be realized by defining a factor node to cover each of these pairwise relationships (see figure 5.1). An alternative is shown in figure 5.3 where a single type of factor $\psi_{\{Y_t, Y_{t-1}, \mathbf{X}_t\}}$ covers pairwise and feature/label dependencies at once. This notation simplifies the derivations in the following section and nevertheless defines the same family of distributions if the joint factor is the product of the two individual

[1] In these publications the base functions are called *features* or *feature functions*. A feature is nothing but a function which "processes" raw sensor data and returns a value which is suitable for the learning algorithm. Therefore it makes perfect sense to talk about features if the input values $\mathbf{x}$ to the CRF is the raw data. Here however, we assume that the input values $\mathbf{x}$ already are the output of the feature functions, and hence the base functions indeed add a new layer to the model.

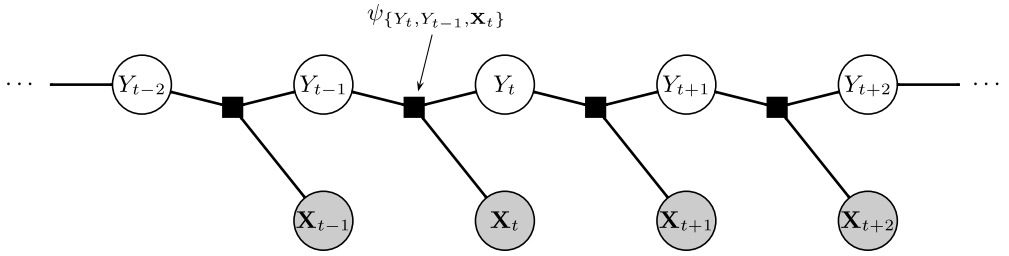

Figure 5.3: An alternative form for linear chain CRFs.

factors of figure 5.1:

$$\psi_{\{Y_t,Y_{t-1},\mathbf{X}_t\}}(y_t, y_{t-1}, \mathbf{x}_t) \stackrel{\text{def}}{=} \psi_{\{Y_t,Y_{t-1}\}}(y_t, y_{t-1}) \cdot \psi_{\{Y_t,\mathbf{X}_t\}}(y_t, \mathbf{x}_t).$$

Linear chain CRFs are often used for temporal models and so $\mathbf{X}_t$ and Y_t then stand for the observation and respective label at time t. Note however that we will also use the index t to identify random variables even if there may be no temporal interpretation.

The second important aspect about the factor nodes are the functions they represent. The more flexible they are the more complex relationships between the connected variables can be learned. On the other hand, most applications heavily rely on parameter tying between the factors, i. e. the factors are grouped into C sets Ψ_c, the *factor templates*, and all factors of a template represent the same compatibility function:

$$\psi_A(\mathbf{x}_{\langle A\rangle}, \mathbf{y}_{\langle A\rangle}) = \psi_c(\mathbf{x}_{\langle A\rangle}, \mathbf{y}_{\langle A\rangle}) \quad \text{for all } \psi_A \in \Psi_c. \tag{5.4}$$

This captures that certain random variables of the model share the same dependencies and therefore only one compatibility function for these groups needs to be learned. In linear chain CRFs, for example, it is commonly assumed that the random variable pairs (Y_{t-1}, Y_t) and (Y_t, Y_{t+1}) have the same dependency because the underlying stochastic process doesn't change over time. So typically in figure 5.1 one would define two factor templates[2] and only one in figure 5.3.

Apart from greatly reducing the number of parameters to be learned, parameter tying also allows to train one CRF and then infer labels in another CRF with the same factor templates but a different graph topology. There are limitations to this, but obviously the parameters learned for a spatial context sequence can be applied to longer or shorter sequences. This also applies to the training data: instead of having several instantiations

[2]Irrespective of the semantic interpretation, it should be clear that the factors $F_{\{Y_t,\mathbf{X}_t\}}$ and $F_{\{Y_t,Y_{t-1}\}}$ can't belong to the same template, because they have incompatible domains. The other way around, however, it would for example be valid to divide the factors $F_{\{Y_t,Y_{t-1}\}}$ into more than one template set.

of the random variables of a CRF, one assumes without loss of generality to only have one instantiation for a larger CRF which consists of several disconnected copies of the original graph. The derivation of virtual evidence boosting (section 5.4) will hence assume that the training data is given as a single sample.[3]

5.3 Belief Propagation for Inference in CRFs

The conditional probability for a full set of labels $\mathbf{y}$ given the features $\mathbf{x}$ can be directly computed with formula (5.1) (assuming that $Z(\mathbf{x})$ is known or only ratios of probabilities are sought after). However for training CRFs with virtual evidence boosting, marginal probabilities of the form $\mathrm{P}(Y_t \mid \mathbf{X})$ are needed.

Furthermore, in the application of a trained CRF for predicting labels, the most likely label vector

$$\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathrm{P}(\mathbf{Y} = \mathbf{y}' \mid \mathbf{X} = \mathbf{x}') \tag{5.5}$$

for a previously unseen vector of input values $\mathbf{x}'$ has to be found. For singly connected CRFs, i. e. graphs that that do not contain loops, the most probable label assignment can be efficiently found using the so-called *Viterbi algorithm* [e. g. Sutton and McCallum, 2006]. Otherwise, an assignment y' can be chosen by maximizing the marginal probabilities of its components:

$$\mathbf{y}' = \arg\max_{\mathbf{y}'} \prod_t \mathrm{P}(Y_t = y'_t \mid \mathbf{X} = \mathbf{x}'). \tag{5.6}$$

While examples can be constructed where the approximation (5.6) yields suboptimal label assignments, i. e. the assignment $\mathbf{y}'$ does not maximize the joint conditional probability as in (5.5), this approach still yields reasonable predictions. All experiments in section 6.3 use equation (5.6) for testing the classification performance of learned models.

The process of computing marginal probabilities is called *inference*. It is clear that for any non-trivial application the marginals can't be computed by simply summing over all possible assignments of the non-fixed variables. However by storing intermediate sums, the runtime can be reduced significantly, in many cases (cf. section 5.3.3) from exponential in the number of label variables to linear. For linear chain CRFs this yields the *forward-backward algorithm* which can be generalized to *belief propagation* and *loopy belief propagation* for general topologies. The dynamic programming idea behind these algorithms shall be first illustrated through a derivation of forward-backward (closely

[3]The empirical expectation which will occur in those formulas is therefore a "sum" over that one sample.

following Sutton and McCallum [2006]), before discussing how this is applied to general graphs.

5.3.1 The Forward-Backward Algorithm

As discussed in the previous section (see figure 5.3), a linear chain CRF contains the sequence of factors $\psi_t \stackrel{\text{def}}{=} \psi_{\{Y_t, Y_{t-1}, \mathbf{X}_t\}}$ and one factor $\psi_1 \stackrel{\text{def}}{=} \psi_{\{Y_1, \mathbf{X}_1\}}$ where the predecessor label node doesn't exist. (For the sake of a shorter notation, $\psi_1(y_1, y_0, \mathbf{x}_1) \stackrel{\text{def}}{=} \psi_1(y_1, \mathbf{x}_1)$ will implicitly occur in the following formulae.) Using distributive law, the normalization $Z(\mathbf{x})$ (cf. equation 5.2) can be written as

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \tag{5.7a}$$

$$= \sum_{y_T} \sum_{y_{T-1}} \psi_T(y_T, y_{T-1}, \mathbf{x}_T) \sum_{y_{T-2}} \psi_T(y_{T-1}, y_{T-2}, \mathbf{x}_{T-1}) \sum_{y_{T-3}} \dots \tag{5.7b}$$

Careful investigation of the indices shows that the inner sums only depend on the variable of the immediate enclosing sum. Instead of naively recomputing inner sums for each outer summation, an exponential amount of time can be saved by applying a caching strategy.

This leads to the definition of a set of *forward variables* α_t, each of which stores the value of an inner sum

$$\alpha_t[j] \stackrel{\text{def}}{=} \sum_{y_1, \dots, y_{t-1}} \prod_{t'=1}^{t} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \Big|_{y_t = j} \tag{5.8a}$$

$$= \sum_{y_{t-1}} \psi_t(j, y_{t-1}, \mathbf{x}_t) \sum_{y_{t-2}} \psi_T(y_{t-1}, y_{t-2}, \mathbf{x}_{t-1}) \sum_{y_{T-3}} \dots \tag{5.8b}$$

for each possible value j of the enclosing sum's variable y_t. The forward variables α_t are vectors of length $|\text{dom}(y_t)|$ and $\alpha_t[j]$ denotes the j-th entry of that vector. The forward variables can be efficiently be computed by the recursion

$$\alpha_t[j] = \sum_{y_{t-1}} \psi_t(j, y_{t-1}, \mathbf{x}_t) \cdot \alpha_{t-1}[y_{t-1}] \tag{5.9}$$

with the initialization $\alpha_1[j] = \psi(j, \mathbf{x}_1)$. The equivalence of (5.8) with the recursive definition (5.9) of the forward variables can be easily shown by induction. With the forward variables equation (5.7) can be simplified to $Z(\mathbf{x}) = \sum_j \alpha_T[j]$.

The backward recursion is using exactly the same approach, with the exception that the sums in (5.7a) are pushed in in the reverse order, resulting in an outermost sum over

y_1 and so on. The backward variables are defined as

$$\beta_t(j) = \sum_{y_{t+1},\ldots,y_T} \prod_{t'=t+1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'})\Big|_{y_t=j} \tag{5.10}$$

and recursively computed through

$$\beta_t(j) = \sum_{y_{t+1}} \psi_{t+1}(y_{t+1}, j, \mathbf{x}_{t+1}) \cdot \beta_{t+1}[y_{t+1}] \tag{5.11}$$

with the initialization $\beta_T[j] = 1$.

So far the forward and backward recursions only each provide a way of computing the normalization $Z(\mathbf{x})$. However by combining the results from both, the marginal distributions for each label variable can be determined. By definition the joint probability (see equation 5.1) can be marginalized w. r. t. to a variable Y_t by summing over all assignments of the remaining variables. Again by applying the distributive law, we see that the sums can be grouped and eventually be replaced by the forward and backward variables:

$$\begin{aligned} \mathrm{P}(Y_t = y_t \,|\, \mathbf{X} = \mathbf{x}) &= \frac{1}{Z(\mathbf{x})} \left(\sum_{y_1,\ldots,y_{t-1}} \prod_{t'=1}^{t} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \\ &\qquad \left(\sum_{y_{t+1},\ldots,y_T} \prod_{t'=t+1}^{T} \psi_{t'}(y_{t'}, y_{t'-1}, \mathbf{x}_{t'}) \right) \\ &= \frac{1}{Z(\mathbf{x})} \, \alpha_t[y_t] \, \beta_t[y_t]. \end{aligned} \tag{5.12}$$

Analogously, the marginal probability over two neighboring label variables

$$\mathrm{P}(y_{t-1}, y_t \,|\, \mathbf{x}) \propto \alpha_{t-1}[y_{t-1}] \, \psi_t(y_t, y_{t-1}, \mathbf{x}_t) \, \beta_t[y_t] \tag{5.13}$$

can be computed easily. Since the probabilities have to sum to 1 (and ensuring this feasible for the marginals), it is common to omit normalization factors and indicate this by replacing equalities by the "proportional to" sign "$\propto$".

5.3.2 Belief Propagation in Chains and Trees

For the generalization of forward-backward to belief propagation, we'll first consider the same problem, inference in linear chain CRFs, but using the notation and ideas of belief propagation. The forward and backward variables are called *messages* in belief propagation. The symbol $m_{A\to t}$ denotes the message which is passed from the factor node

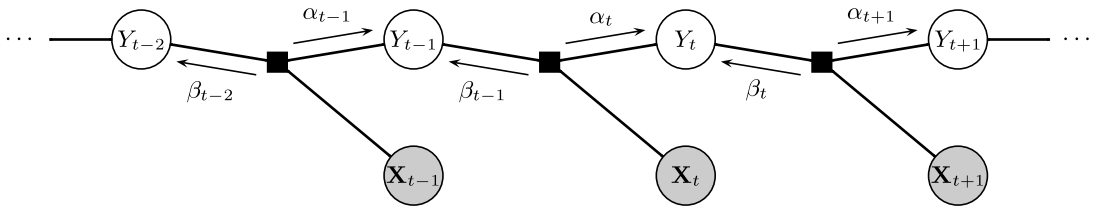

Figure 5.4: Forward and backward variables shown as belief propagation messages.

ψ_A to the variable node Y_t. Using the abbreviation $A_t \stackrel{\text{def}}{=} \{Y_t, Y_{t-1}, \mathbf{X}_t\}$, the forward and backward variables are equivalent to the following messages (cf. figure 5.4):

$$m_{A_t \to t} = \alpha_t \tag{5.14}$$

$$m_{A_{t+1} \to t} = \beta_t \,. \tag{5.15}$$

An idea which is often presented in introductions to belief propagation [e. g. Yedidia et al., 2002] is that the messages contain what the the sender and the network behind it believe that the receiving variable Y_t should be. This is expressed as a multinomial distribution, i. e. each component of the message vector is proportional to how likely the each of the possible values of the receiving random variable is. Consequently, the combined *belief* about a variable Y_t, the marginal probability $\mathrm{P}(Y_t = y_t \,|\, \mathbf{X} = \mathbf{x})$, is the product of all incoming messages at that node (cf. equation 5.12 and 5.18).

The informal definition of the recursion for computing messages is as follows: the message $m_{A \to t}$ combines (multiplies) the information from

1. the compatibility of assignments to the neighbors of the factor node ψ_A and
2. the messages which were received from the other neighbors of ψ_A, with the exception of the message sent from ψ_A itself.

In the case of a linear chain there is only one other message and so this mirrors the computations from the forward and backward iterations (5.9) and (5.11).

Instead of talking about believes and information, a different view may be more helpful to understand belief propagation. One can look at what global formula each of the messages stands for and how these are useful for the computation we want to perform. The conditional probabilities (5.1) are computed by taking the product of all factors. For the marginal probabilities these products are summed over all combinations of values for all but one variable. The messages (cf. definition of the forward variables (5.8) and figure 5.4) contain the product over all factors that lie behind the arc the message is passed along, i. e. for a message $m_{\{Y_t, Y_{t-1}, \mathbf{X}_t\} \to t}$ these are the factors $\{\psi_1, \ldots, \psi_t\}$, and these

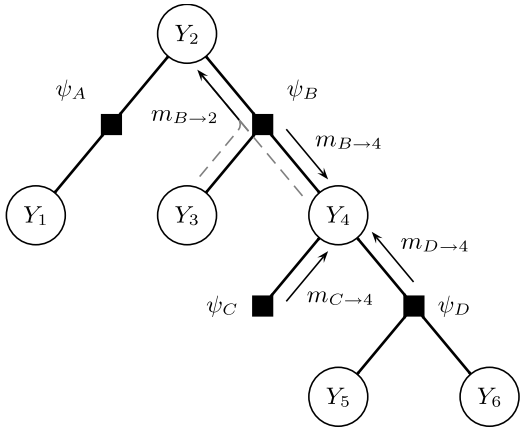

Figure 5.5: An example for a loop-free CRF.

products are summed over all variables except the one where the message is sent to. Since the value y_t is needed for determining the value of the factor ψ_t, the message is a vector with the result for every possible value y_t.

Now the calculations of marginal probabilities for general tree-shaped CRFs (figure 5.5) are straightforward. For example the marginal distribution of Y_4 is the product over all factors, summed over all variables except Y_4. Similar to (5.12) the sums can be grouped into three groups which cover the three subgraphs that the node Y_4 partitions the graph into. Again, each of the groups of sums is equal to one of the messages which arrives at the node Y_4 because they each have exactly one of the subgraphs behind them. Therefore the marginal probability for $Y_4 = j$ is the product of the j-th components of the messages:

$$\mathrm{P}(Y_4 = j \mid \mathbf{X} = \mathbf{x}) = m_{B\to 4}[j]\, m_{C\to 4}[j]\, m_{D\to 4}[j]$$

Next we will show how messages are computed recursively at the example of the message from the factor node ψ_B to the variable node Y_2 (see figure 5.5). According to our definition it has to contain the sum-product over the variables and factors behind it. First of all this means the factor ψ_B itself, and for the factors beyond we already know that they are included in the messages $m_{C\to 4}$ and $m_{D\to 4}$. Also, additionally to what is marginalized out in those messages, we need to sum over the nodes Y_3 and Y_4, which are the nodes connected to ψ_B other than Y_2. The example message therefore is calculated through

$$m_{B\to 2}[j] = \sum_{y_3} \sum_{y_4} \psi_B(j, y_3, y_4)\, m_{C\to 4}[y_4]\, m_{D\to 4}[y_4]\,.$$

The message $m_{B\to 4}$ is obviously not included in the recursion because it covers a part of the graph which doesn't lie behind the arc from ψ_B to Y_2.

In general, belief propagation specifies the following recursive formula for computing the messages [e. g. Yedidia et al., 2005]

$$m_{A\to t}[j] = \sum_{\mathbf{y}_{\langle A\setminus\{Y_t\}\rangle}} \psi_A(\mathbf{x}_{\langle A\rangle}, \mathbf{y}_{\langle A\rangle})\big|_{y_t=j} \prod_{Y_{t'}\in A\setminus\{Y_t\}} m_{t'\to A}[y_{t'}], \tag{5.16}$$

where the $m_{t'\to A}$ can be seen as the intermediate messages from the variable nodes $Y_{t'}$ to the factor ψ_A

$$m_{t'\to A}[y_{t'}] = \prod_{\psi_{A'}\in \mathrm{nb}(Y_{t'})\setminus\{\psi_A\}} m_{A'\to t'}[y_{t'}] \tag{5.17}$$

and $\mathrm{nb}(Y_{t'}) \stackrel{\mathrm{def}}{=} \left\{\psi_{A'}\in\Psi \,|\, Y_{t'}\in A'\right\}$ is the set of all neighbors of the node $Y_{t'}$. Recall that the indices of the factor nodes ψ_A are the set of variable nodes Y_t they are connected to. The marginals for a node are proportional the the product of the incoming messages

$$\mathrm{P}(Y_t = j \,|\, \mathbf{X}=\mathbf{x}) \propto \prod_{\psi_A\in\mathrm{nb}(Y_t)} m_{A\to t}[j]. \tag{5.18}$$

It is easy to see that in a dynamic programming manner and starting from the leaves, every message in the graph can be eventually be calculated. A formal proof would perform an induction over the number of factor nodes behind the arc of the message, but this would not give any more insight than the illustrative arguments developed here earlier.

5.3.3 Loopy Belief Propagation

The dynamic programming argument is however not true if the graph contains a loop. Every message of the loop is behind itself and so would need to be known for being computed. This contradiction can be broken by simply initializing the messages with a uniform distribution, i. e. the same positive value for each of the vector components, and then repeat the message update (5.16) for all messages in any order until they converge. If the graph does not contain any loops, this indeed yields the same exact results as before. Otherwise however this is neither guaranteed to yield accurate marginal distributions, nor even to converge. Nevertheless, loopy belief propagation has been used successfully in many applications. The interested reader is referred to the work by Yedidia et al. [2005], who show the equivalence of the algorithm to the *Bethe approximation to the free energy* and how improvements of that approximation lead to the more accurate *generalized belief propagation*. The second issue with loopy belief propagation in comparison to belief propagation in chains and trees is the computational complexity. Instead of linear time with respect to the number of nodes, it may take many iterations over all nodes until

the messages converge. VEB for for training conditional random fields uses loopy belief propagation although it may be possible include other inference methods if necessary.

5.4 Training CRFs with Virtual Evidence Boosting

As discussed in the related work section 2.2, there are several issues with the standard training of conditional random fields by maximum likelihood (ML) [Sutton and McCallum, 2006]. Each learning iteration requires running inference, for example loopy belief propagation, and even with modern optimization methods, tens to a few hundred iterations are required. Furthermore there is no native support for continuous feature values.

Therefore, a new method for training CRFs called virtual evidence boosting (VEB) has been recently published by Liao, Choudhury, Fox, and Kautz [2007], with more details in the PhD thesis of the first author [Liao, 2006]. It applies the optimization strategy of LogitBoost (see chapter 4) which leads to many advantages over maximum likelihood estimation and other methods (cf. related work in section 2.2).

- VEB learns the dependencies of the labels on input values as well as correlations between the labels in a unified framework. Nevertheless the neighboring label variables are not simply treated as observed and so the dependencies amongst labels are not overestimated.

- Also, VEB is able to handle both discrete and continuous observations. It reduces the learning iteration to a weighted least squares problem and then employs standard function approximators to represent the update.

- The parameter estimation process is based on Newton's method for optimization, and so VEB training takes less iterations than gradient descent methods not using the second derivative. 10–20 iterations are sufficient for most applications. Still, each iteration can be brought to the same computational complexity as ML.

- VEB is able to perform *feature selection* and detect dependencies between the variables, while ignoring irrelevant features and connections in the graph. In each iteration, only those factors are updated which yield the best improvement of the optimization criterion so that only significant features and dependencies between the labels are included in the model.

- VEB can in theory be used to train conditional random fields of almost arbitrary topology, assuming that belief propagation converges and infers accurate marginal

probabilities. A limitation applies to the number of variables a factor can be connected to (see section 5.4.1). Also there may be some issues on highly complex topologies in practice (see section 6.3.3).

In order to achieve these advantages, VEB contains several approximations. First of all it maximizes the *per-label likelihood*

$$\hat{\mathrm{E}}\big[\log \prod_{t=1}^{T} \mathrm{P}(Y_t \mid \mathbf{X})\big] \tag{5.19}$$

instead of the joint likelihood of all labels

$$\hat{\mathrm{E}}\big[\log \mathrm{P}(\mathbf{Y} \mid \mathbf{X})\big]. \tag{5.20}$$

Additionally it assumes that the factors which are not immediate neighbors of each node are fixed during an individual training iteration, so the Newton steps are actually computed with respect to an approximation of the per-label likelihood (see section 5.4.2).

The local view onto small neighborhoods in the CRF is based on a similar idea from *maximum pseudo-likelihood* estimation (MPL) [Besag, 1975]. The pseudo-likelihood is probability of a label variable, given all connected variable nodes, the so-called *Markov blanket*, including other label variables

$$\hat{\mathrm{E}}\big[\log \prod_{t=1}^{T} \mathrm{P}(Y_t \mid \mathrm{nbv}(Y_t), \mathbf{X})\big] \tag{5.21}$$

Learning the parameter by maximizing the pseudo-likelihood can be done easily, because it doesn't require any inference. However it has been observed, that MPL overestimates the dependency between the labels [Geyer and Thompson, 1992]. To comprehend this issue, imagine a linear chain CRF where the true label only switches a few times. MPL then may only learn the few situations, i. e. input value combinations, where the label is not the same as the neighbors' labels. This means that the learned model may infer long sequences of labels only based on the presence of a few unique feature values which have been the "switching cues" in the training data. It is easy to imagine, that this may be very unstable and easily leads to bad generalization performance.

In VEB the learning algorithm is not given the true labels of the neighboring hidden nodes but only knows a probability distribution over the labels of those nodes. The distribution reflects what currently can be inferred with the model parameters. This leads to the following behavior: During the first iterations VEB selects the factors which are connected to the input nodes, which is equivalent to learning the labels indepen-

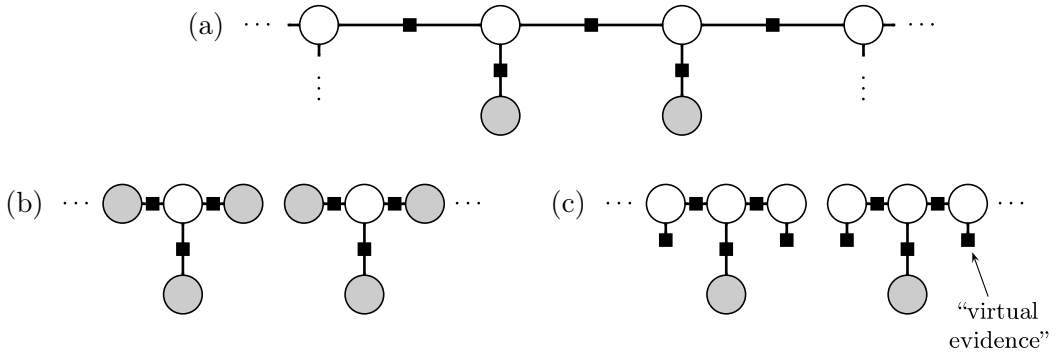

Figure 5.6: Example of the network transformation schemes of maximum pseudo-likelihood and virtual evidence boosting. (a) Original CRF. (b) Transformation for MPL. (c) Transformation idea for VEB (not applied in following derivation).

dently. Once the model is able to produce reasonably confident estimates for for the label variables, dependencies between the variables may included in the model. This is inherent in the strategy of always picking the factor whose parameter update yields the largest improvement of the per-label likelihood, so there is no need for distinct phases of learning feature and neighbor dependencies. This conception of VEB is overcoming the bootstrapping problem that may occur if a graphical model was trained with MPL: the correct labels of neighboring variables need to be known to correctly infer a label.

The MPL optimization criterion (5.21) can also be seen as maximum likelihood on a transformed conditional random field: All connections between hidden nodes are removed and each of the affected nodes gets a new connection to a copy of its former neighbor (see figure 5.6 a-b). The copied neighbor nodes are treated as observed, i. e. their label is fixed to the true label. In the graphical modeling context, this information which is inserted into the graph is called *evidence* [e. g. Korb and Nicholson, 2004]. The resulting CRF then only has small, independent patches with one hidden variable each. VEB training can be seen in a similar way except that the neighboring labels are not considered to be known exactly. Instead they are treated as *virtual evidence* [Pearl, 1988], i. e. they are given a prior distribution which is set according to the inference in the original conditional random field (see figure 5.6c). The view of VEB as a network transformation scheme with virtual evidence, which has coined the name of the algorithm, is promoted by the original authors [Liao et al., 2007]. Here however, the derivation will be based on the equivalent in the original, unmodified CRF.

In the now following sections, the formulas and logic of VEB will be derived, implementing the ideas outlined above. The first section illustrates why we can assume that the CRFs only have two types of factors, pairwise factors and local factors, and defines

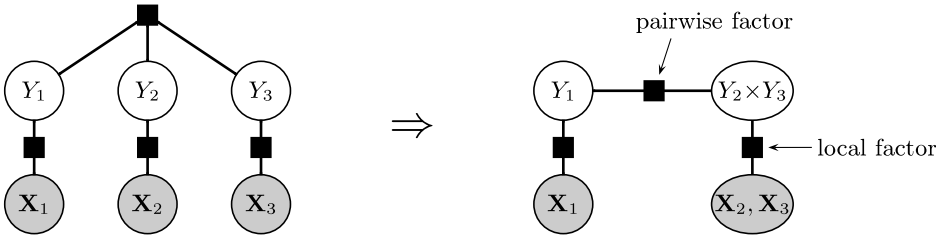

Figure 5.7: Scheme to convert factors connected to more than two hidden nodes to pairwise factors.

the parameterizations of their factor functions. The second section investigates the optimization criterion of VEB and discusses the approximations which allow to optimize it by Newton's method. In the third section it is shown that a quasi-Newton update to the parameters can be found by solving a weighted least squares problem, and subsections detail the formulae for each factor type. There are two versions for computing the updates for pairwise factors with different computational complexity which are presented in section 5.4.3 (iv). Before listing the complete algorithm in the fifth section, section 5.4.4 illustrates how feature selection is generalized to CRFs by only applying parameter updates to some factors.

The derivation is a complete redevelopment done in the preparation of this Master's thesis. It is based on the re-derivation of LogitBoost (section 4), so the reader is encouraged to also read the proofs of theorem 4.1 and 4.7. Liao [2006] on the other hand extends the the original derivation of LogitBoost [Friedman et al., 2000] and fails to see a difference in the application to CRFs which results in an error. Therefore both our update formulae for pairwise factors differ from the one by Liao et al. [2007]. They are compared in the sixth section compares and it is discussed why their formula may still work in practice. In fact, Liao's VEB generally surpasses the performance of our versions of VEB. The results of an empirical comparison can be found in section 6.3.2.

5.4.1 Parameterization of Local and Pairwise Factors

From now on we will assume that the CRF only contains two types of factors: *pairwise factors* which connect two hidden variable nodes, and *local factors* which connect one hidden node with one or several input variable nodes (cf. figure 5.7). This is not a very strong limitation because every factor graph can be transformed into an equivalent graph only containing these factor types, and in most cases this graph can be used instead. A possible strategy for the transformation is to join label nodes which are connected to factors not satisfying the criteria (see figure 5.7). The joined nodes then

stand for a random variable over the cross-product of the domains of the joined variables. This step can be repeated until every factor is either reduced to a pairwise or a local factor. The practical limitation to this procedure is caused by the fact that the joined nodes have a domain which is exponential in the number joined nodes. So training the transformed CRF may become infeasible if the original graph contains factors which connect large numbers or even all hidden variables. (The same problem arises with the strategy described in [Yedidia et al., 2002].) This situation is generally hard to handle for a learning algorithm and is best addressed by customized algorithms [Liao et al., 2005, Liao, 2006]. To reiterate, the only limitation on the topology imposed by VEB is that each factor only connects a small number of hidden nodes.

For notational convenience we will also state that every hidden node has at most one local factor connected to it. This is no limitation because there may be a vector of observations connected to the other end of that factor. That random vector node $\mathbf{X}$ may be seen as an abbreviation for several observed nodes X_i (cf. figure 5.2 on page 36).

The factor functions themselves are learned from a very flexible class of functions. In fact, VEB requires that the factors have a separate set of parameters for every label assignment of the connected hidden nodes. This seemingly complicated concept is very simple in practice. For the local factors there is a separate model function $F_{A,j}(\mathbf{x})$ for every possible label j of the hidden variable. The local factor function "selects" the respective model function according to the label, so these factors take the form

$$\begin{aligned}\psi_{\{Y_t,\mathbf{X}_t\}}(y_t,\mathbf{x}_t) &= \exp\{F_{\{Y_t,\mathbf{X}_t\}}(y_t,\mathbf{x}_t)\} \\ &= \exp\Big\{\sum_{j\in\mathrm{dom}(Y_t)} \mathbf{1}_{\langle y_t=j\rangle}\cdot F_{\{Y_t,\mathbf{X}_t\},j}(\mathbf{x}_t)\Big\}.\end{aligned} \tag{5.22}$$

With this parameterization, it is easy to verify that the the conditional probability (cf. equation (5.1)) of the CRF shown in figure 5.2 is exactly the symmetric multiple logistic transformation (4.12) from the multiclass version of LogitBoost (section 4.4). Just like in LogitBoost, the model functions $F_{A,j}(\mathbf{x})$ are sums of any base functions that can be the output of a function approximator.

For the pairwise factors, the parameter separation requirement of VEB means that they need to be arbitrary functions over the discrete domains of the connected variables $\psi_{\{Y_t,Y_{t'}\}} : \mathrm{dom}(Y_t)\times\mathrm{dom}(Y_{t'})\mapsto\mathbb{R}^+$. They can only take a finite number of values and hence can be parameterized by matrices $F_{\{Y_t,Y_{t'}\}}$ which store the logarithms of the

compatibilities:

$$\begin{aligned}\psi_{\{Y_t,Y_{t'}\}}(y_t, y_{t'}) &= \exp\Big\{\sum_{j,k} \mathbf{1}_{\langle y_t=j\rangle}\, \mathbf{1}_{\langle y_{t'}=k\rangle} \cdot F_{\{Y_t,Y_{t'}\}}[j,k]\Big\} \\ &= \exp\{F_{\{Y_t,Y_{t'}\}}[y_t, y_{t'}]\}.\end{aligned} \tag{5.23}$$

Recall that the square bracket arguments have a role like subscripts in common notation, i. e. here they select the respective element of the matrix.

In order to make the parameterization of the factor functions unique (cf. remark 4.4), a centering constraint for each factor is introduced. They require that the model functions and the elements of the model matrix of a factor each sum to zero:

$$\sum_j F_{\{Y_t,\mathbf{X}_t\},j}(\mathbf{x}_t) = 0 \quad \text{for all } \mathbf{x}_t \in \operatorname{dom}(\mathbf{X}_t) \tag{5.24}$$

$$\sum_{j,k} F_{\{Y_t,Y_{t'}\}}[j,k] = 0. \tag{5.25}$$

This constraint has to be taken into account when optimizing a factor's parameters. After a learning update to one of the parameters, it is simple to re-center the parameterization because adding the same fixed value or function to *all* parameters of a factor doesn't change the represented probabilities (cf. equations (5.1)–(5.3), the constant summands in the exponents cancel out). However, the centering constraints also introduce the issue that there are more parameters than actual degrees of freedom. In analogy to the multiclass version of LogitBoost (section 4.4) we will ignore this issue at first and compute updates for all parameters, but then discount the updates to make them optimal (again with respect to quasi-Newton steps computed for all possible non-symmetric parameterizations, see theorem 4.7). VEB updates each of the matrix components separately, respectively each of the model functions (see section 5.4.3). In both cases one of these entities should have been bound by the centering constraint, so the updates need to be discounted by one over the number of separate updates. In a nutshell, the updates to the model matrix $F_{\{Y_t,Y_{t'}\}}$ of a pairwise factor are multiplied by $\big(1-1/|\operatorname{dom}(Y_t) \times \operatorname{dom}(Y_{t'})|\big)$ and the updates to the model functions $F_{\{Y_t,\mathbf{X}_t\},j}$ of a local factor are multiplied by $\big(1-1/|\operatorname{dom} Y_t|\big)$.

This concludes the definition of the parameterization of the factor functions used by virtual evidence boosting. In particular the local factors (5.22) are much more flexible than those in CRFs trained by maximum likelihood because the base functions are not fixed by also learned by "weak classifiers". Interesting is however that VEB can't be limited to learning factor functions from a smaller class of functions. This are for example

factor functions of the form

$$\psi(y_1, y_2) = \exp\{\alpha d_1(y_1, y_2) + \beta d_2(y_1, y_2)\}.$$

The idea behind this example may be that there are two predefined meaningful distance metrics d_1 and d_2 and only their weights α and β have to be learned. Such an approach may be preferable in an application where the number of labels is large and hence the number of parameters of arbitrary pairwise factors (5.23), which is quadratic in the number of labels, may be excessive. Developing VEB for constrained pairwise factors is a task for future research.

5.4.2 Optimization Criterion and Approximations

In this section we will investigate the optimization criterion of VEB and the approximations which allow tho apply Newton's quickly converging method for optimization.

VEB maximizes the the per-label likelihood (5.19), which can be written as a sum over training samples and label nodes

$$\frac{1}{S} \sum_{s=1}^{S} \sum_{t=1}^{T} \log \mathrm{P}(Y_t = y_t^{(s)} \mid \mathbf{X} = \mathbf{x}^{(s)}, F_{\langle\Psi\rangle}). \tag{5.26}$$

This term is a function of the parameters of the CRF, which is illustrated by the adding $F_{\langle\Psi\rangle}$ to the right side of the conditional probability. The parameters are the "variable" of the optimization process, so they are iteratively adapted with the aim to maximize the optimization criterion. In each iteration VEB computes an additive update to the current parameters of all factors $F_{\langle\Psi\rangle}$: a matrix f_A is added to the model matrix of each pairwise factor ψ_A and base functions $f_{B,j}$ are added to the model functions of each local factor ψ_B.[4] Hence, within an iteration the current parameter set $F_{\langle\Psi\rangle}$ is considered fixed, and an update $f_{\langle\Psi\rangle}$ (which abbreviates all of the above f_A and $f_{B,j}$; see also the notation definitions in section 3.1) needs to be found to maximize

$$\hat{\mathrm{E}}\big[\sum_{t=1}^{T} \log \mathrm{P}(Y_t \mid \mathbf{X}, F_{\langle\Psi\rangle} + f_{\langle\Psi\rangle})\big]. \tag{5.27}$$

It is in general not possible to maximize this term exactly. What we are aiming for

[4] For the pairwise factors the additive update is equivalent to a full update which would allow an arbitrary parameter set $F'_{\langle\Psi\rangle}$ to be the output of an iteration. For local factors the additive update parallels LogitBoost (see section 4.1), which is strictly less than a full update: each of the factors' model functions $F_{B,j}$ is a sum of base functions, and only one further base function is added, leaving the other base functions unchanged.

is a quasi-Newton step which effectively does the following: The optimization criterion is approximated by a 2^{nd} order Taylor polynomial and then the best "value" for $f_{\langle\Psi\rangle}$ is determined analytically with respect to that approximation. The convenience of this approach lies in the fact that the approximation is only based on information about the target function at the current parameter set $F_{\langle\Psi\rangle}+0$. This is important because even just evaluating the term (5.27) with any nonzero update $f_{\langle\Psi\rangle}$ would require to run inference on the whole CRF again. Still, the "information" required about the optimization criterion is the first and second derivative with respect to $f_{\langle\Psi\rangle}$, and unfortunately computing the latter is not feasible.

Therefore the Newton step is taken with respect to an approximation of (5.27) which drops some of the dependencies on the update $f_{\langle\Psi\rangle}$. The approximation assumes that each of the marginal probabilities $\mathrm{P}(Y_t \mid \mathbf{X})$ are functions only of the updates to the respective immediate neighboring factors (cf. definition of $\mathrm{nb}(Y_t)$ near equation (5.17)), with the remaining factors being fixed. This yields the *update criterion*

$$p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle}) \stackrel{\text{def}}{=} \hat{\mathrm{E}}\big[\sum_{t=1}^{T} \log \mathrm{P}(Y_t \mid \mathbf{X}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})\big] \tag{5.28}$$

for which a Newton step can indeed be computed (see following section). We introduce the term *label probabilities* to refer to the marginal probabilities with partially updated parameters from the update criterion. Note that despite the partial update within the label probability formulae, the CRF's parameters will be set to $F_{\langle\Psi\rangle} + f_{\langle\Psi\rangle}$ at the end of the iteration. Therefore the value of the optimization criterion will be given by (5.27), although $f_{\langle\Psi\rangle}$ itself is the optimum of the Taylor polynomial of an approximation of the original criterion.

From the computational point of view, the key idea behind the update criterion is that it can be computed from existing belief propagation (BP) messages and no new and costly inference is required. This will be exemplified for the simple CRF in figure 5.8 in the following paragraph before writing down the general formula in equation (5.31).

Example 5.2. At the beginning of each iteration, VEB runs loopy belief propagation on the CRF with the current parameter set $F_{\langle\Psi\rangle}$. From this the label probabilities can be computed by multiplying all incoming messages at the respective node (cf. equation 5.18), for example

$$\mathrm{P}(Y_1 = y_1^* \mid \mathbf{x}, F_{\langle\Psi\rangle}) = \frac{m_{B\to 1}[y_1^*] \cdot m_{A\to 1}[y_1^*]}{\sum_{y_1} m_{B\to 1}[y_1] \cdot m_{A\to 1}[y_1]} \tag{5.29}$$

To make the dependency on the neighboring factor functions explicit, the iterative for-

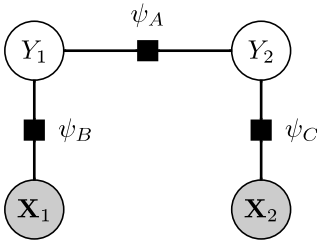

Figure 5.8: A simple CRF with the pairwise factor ψ_A and the local factors ψ_B and ψ_C. The indices of the factors are the sets of the connected variable nodes, so $A = \{Y_1, Y_2\}$, $B = \{Y_1, \mathbf{X}_1\}$, and $C = \{Y_2, \mathbf{X}_2\}$

mula for the BP messages (5.17) is unrolled once, yielding

$$\begin{aligned} \mathrm{P}(Y_1 = y_1^* \,|\, \mathbf{x}, F_{\langle\Psi\rangle}) &= \frac{\psi_B(y_1^*, \mathbf{x}_1) \cdot \sum_{y_2} \psi_A(y_1^*, y_2)\, m_{2\to A}[y_2]}{\sum_{y_1} \big(\psi_B(y_1, \mathbf{x}_1) \cdot \sum_{y_2} \psi_A(y_1, y_2)\, m_{2\to A}[y_2]\big)} \\ &= \frac{\sum_{y_2} \exp\{F_B(y_1^*, \mathbf{x}_1) + F_A[y_1^*, y_2] + n_{2\to A}[y_2]\}}{\sum_{y_1, y_2} \exp\{F_B(y_1, \mathbf{x}_1) + F_A[y_1, y_2] + n_{2\to A}[y_2]\}} \end{aligned} \tag{5.30}$$

with the definition of log messages $n_{2\to A}[y_2] \stackrel{\text{def}}{=} \log m_{2\to A}[y_2]$ for notational convenience. Since all model functions which occur in this unrolled form belong to the immediate neighboring factors, replacing them by the updated functions yields the label probabilities from the update criterion (5.28).[5] So for getting $\mathrm{P}(Y_1 = y_1^* \,|\, \mathbf{X} = \mathbf{x}, F_A + f_A, F_B + f_B, F_C)$, we would replace F_A by $F_A + f_A$ and F_B by $F_B + f_B$ in equation (5.30). The resulting formula is show in (5.45) in a later example.

The same approach of unrolling the belief propagation message recursion yields the general formula for the label probabilities from the update criterion:

$$\begin{aligned} \mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle}) \propto \\ \sum_{\mathbf{y}_{\langle \mathrm{nbv}(Y_t)\rangle}} \exp\Big\{ \sum_{Y_s \in \mathrm{nbv}(Y_t)} F_{\{Y_t, Y_s\}}[y_t^*, y_s] + f_{\{Y_t, Y_s\}}[y_t^*, y_s] + n_{s\to\{Y_s, Y_t\}}[y_s] \Big\} \\ \times \exp\{F(y_t^*, \mathbf{x}_t) + f(y_t^*, \mathbf{x}_t)\} \end{aligned} \tag{5.31}$$

with the sets $\mathrm{nbv}(Y_t)$ describing all the label variables which are connected to the other

[5] In a loop-free CRF all remaining messages in (5.30) don't depend on the neighboring factors, so replacing all model parameters by their updated version yields exactly one of the label probability terms from the update criterion. If however the CRF contains loops, the messages are not necessarily independent of the neighboring factors. They are still used in the formulas for the label probability, so strictly speaking the notation (5.28) is not quite accurate. (The occurrences of $F_{\langle \mathrm{nb}(Y_t)\rangle}$ in the messages in the unrolled form (5.30) are not replaced by $F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}$.)

ends of Y_t's pairwise factors:

$$\mathrm{nbv}(Y_t) = \big\{Y_s \,|\, \exists \psi_A \in \Psi, A = \{Y_t, Y_s\}\big\}. \tag{5.32}$$

Note that the neighbor variable set $\mathrm{nbv}(Y_t)$ is less than the Markov blanket because the latter would also include the observed variable $\mathbf{X}_t$ connected through the local factor. It shall be further pointed out about the notation that the outer sum in (5.31) ranges over all possible assignments to the variables in $\mathrm{nbv}(Y_t)$ (cf. the notation definitions in section 3.1), while the sum in the exponent simply yields three addends for every connected pairwise factor.

The update criterion ignores some of the influence of the parameter update $f_{\langle\Psi\rangle}$ on the per-label likelihood. Consequently, the accuracy of this approximation depends on how strong the dropped influence would have been in the exact formula. We'll again use the CRF in figure 5.8 as an example.

Example 5.3. Previously we have seen that the approximated label probability of the label Y_1 ignores the influence of an update f_C to the local factor ψ_C. Assuming no other updates, the exact label probability would have the following dependency on f_C:

$$\mathrm{P}(Y_1 = y_1^* \,|\, \mathbf{x}, F_C + f_C) \propto \sum_{y_2} \exp\{F_A[y_1^*, y_2]\} \cdot \exp\{F_C(y_2, \mathbf{x}_2) + f_C(y_2, \mathbf{x}_2)\}. \tag{5.33}$$

We see that the influence of f_C is larger the more skewed the distribution $F_A[y_1^*, \cdot]$ is, or the other way round that the influence averages out if the model matrix F_A has uniform values. The latter is the case at the initialization and during the first iterations of VEB because the label interdependencies only slowly become visible to the algorithm after local factors have been updated. Eventually however the model becomes more certain about the distribution of neighboring label nodes, which is expressed as a skewed model matrix F_A, and so the ignored effect of the update f_C may lead to a decrease of Y_1's label likelihood. In larger CRFs than the one considered here these effects can also propagate over several hidden nodes and so make the issue even worse. As a rule of thumb it can be summarized that the update criterion is a fairly good approximation of the per-label likelihood at first, but gets less accurate with the number of learning iterations.

Note that we haven't considered as to why the algorithm would choose an update f_C to the factor ψ_C. Since the latter connects to Y_2, the update term is part of the label probability of Y_2 and also of the update criterion. Furthermore the Newton step is taken with respect to the template parameters, so if the factors ψ_B and ψ_C have tied parameters, VEB would determine f_B without considering the influence this has on Y_1's label probability via ψ_C.

Some readers may criticize that the entire approach of computing quasi-Newton steps for an approximation to the optimization criterion is flawed. It doesn't have to be. A very successful example is Newton's method for optimization by itself, which also computes iterative updates with respect to an approximation. It obviously depends on the quality of the approximation, and as discussed in the previous example the quality decreases as more of the label variables dependencies are learned. Indeed it has been observed in some rare cases that the algorithm diverges eventually but not before reaching a good parameter set (see section 6.3.2). The second important question is the sensitivity of the resulting updates to the approximation in different parts of the parameter space. Newton's method is known to be fairly unstable by itself and it turns out to compute excessively large updates to the pairwise factors. (Whether or not this is also caused by the additional approximation remains unknown.) However this issue can be overcome by limiting the step size (see following section), yielding a stable and successful learning method (see experiments in [Liao et al., 2007] and section 6.3).

Also, it has to be noted that the view of VEB presented here is equivalent to seeing it as maximum likelihood on the decomposed CRF with virtual evidence, which was motivated in the introduction to section 5.4 on page 48. The latter view may be more convincing as to why VEB works in practice, but in our opinion the description developed here is more useful for understanding what the algorithm actually does.

5.4.3 Updating Factors with Newton's Method

In this section it will be shown how the update criterion is optimized by Newton's method. It is well-known that the Newton steps maximize the 2^{nd} order Taylor polynomial, and in the first subsection this shown to be equivalent to solving a weighted least squares problem. The following subsections then fill in the steps that are specific to local and pairwise factors: computing the derivatives which are used in the Taylor polynomial and solving the weighted least squares problems. So following the update for local factors in subsection two is an example for the pairwise factors and finally the general formulae in the fourth subsection.

We have seen that the update criterion $p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle})$ (5.28) sums over log label probabilities which only have the model parameters of their immediate neighboring factors updated to $F_{\langle \text{nb}(Y_t)\rangle} + f_{\langle \text{nb}(Y_t)\rangle}$. To capture the update criterion in a formal notation, we needed to refer to the model functions of individual factors ($F_{\langle \text{nb}(Y_t)\rangle}$ is short for the vector of all model functions F_A of the factors $\psi_A \in \text{nb}(Y_t)$). For the optimization of the update criterion on the other hand, we are only interested in the actually independent parameters. Recall that many CRFs have tied parameters, so all factors which belong

to a factor template Ψ_c share the same model functions F_c. Here, this means that the updates f_c are identical for factors of the same template, but in the update criterion formula they are still only applied to the immediate neighbors and not all factors of a template.

5.4.3 (i) From Newton to Weighted Least Squares Problems

The parameter update steps of Newton's method can be found by optimizing the second order Taylor approximation of the update criterion. This polynomial requires the gradient and Hessian matrix of the update criterion $p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle})$ (5.28) with respect to all factor template updates $(f_1 \dots f_C)$. Since derivation and sum operations can swap places (the empirical expectation is also just a sum, see definition 4.2 on page 27), the derivatives can be reduced to sums

$$\frac{\mathrm{d}\, p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle})}{\mathrm{d}(f_1 \dots f_C)}\Big|_{f_{\langle\Psi\rangle}\equiv 0} = \hat{\mathrm{E}}\big[\sum_{t=1}^{T} \nabla_t(Y_t, \mathbf{X})\big] \tag{5.34}$$

$$\frac{\mathrm{d}^2\, p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle})}{\mathrm{d}(f_1 \dots f_C)}\Big|_{f_{\langle\Psi\rangle}\equiv 0} = \hat{\mathrm{E}}\big[\sum_{t=1}^{T} W_t(Y_t, \mathbf{X})\big] \tag{5.35}$$

over the gradients and Hessians of the label probabilities

$$\nabla_t(y_t, \mathbf{x}) \stackrel{\text{def}}{=} \frac{\mathrm{d}}{\mathrm{d}(f_1 \dots f_C)} \log \mathrm{P}(y_t \,|\, \mathbf{x}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})\,\big|_{f_{\langle\Psi\rangle}\equiv 0} \tag{5.36}$$

$$W_t(y_t, \mathbf{x}) \stackrel{\text{def}}{=} \frac{\mathrm{d}^2}{\mathrm{d}^2(f_1 \dots f_C)} \log \mathrm{P}(y_t \,|\, \mathbf{x}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})\,\big|_{f_{\langle\Psi\rangle}\equiv 0}\,. \tag{5.37}$$

By applying the distributive law, the Taylor approximation therefore becomes

$$p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle}) \approx c + \hat{\mathrm{E}}\big[\sum_{t=1}^{T} (f_1 \dots f_C) \cdot \nabla_t(Y_t, \mathbf{X}) + \frac{1}{2}(f_1 \dots f_C) \cdot W_t(Y_t, \mathbf{X}) \cdot (f_1 \dots f_C)^T\big]. \tag{5.38}$$

The current per-label likelihood $c = p\ell(F_{\langle\Psi\rangle}, 0)$ doesn't depend on any update, so it is not important for finding the optimal f_c. Just like in LogitBoost, only quasi-Newton steps are feasible where all off-diagonal elements of the Hessian matrix are set to zero. This allows to rearrange the right-hand side of equation (5.38) into one weighted squared

error term for each parameter $f_c[i]$:

$$p\ell(F_{\langle\Psi\rangle}, f_{\langle\Psi\rangle}) \approx c - \frac{1}{2}\,\hat{\mathrm{E}}\Big[\sum_{t=1}^{T}\sum_{c=1}^{C}\sum_{i}(-W_t[c,i])\,f_c[i]^2 - 2\nabla_t[c,i]\,f_c[i]\Big] \tag{5.39a}$$

$$= c - \frac{1}{2}\sum_{c=1}^{C}\sum_{i}\sum_{t=1}^{T}\Big(\hat{\mathrm{E}}_{(-W_t[c,i])}\Big[\big(f_c[i] - \frac{\nabla_t[c,i]}{-W_t[c,i]}\big)^2\Big] - \hat{\mathrm{E}}\Big[\frac{\nabla_t[c,i]^2}{-W_t[c,i]}\Big]\Big) \tag{5.39b}$$

where $\nabla_t[c,i]$ is a shorthand for the element of the gradient $\nabla_t(Y_t, \mathbf{X})$ corresponding to the parameter $f_c[i]$, and similarly $W_t[c,i]$ abbreviates the respective diagonal element from $W_t(Y_t, \mathbf{X})$. The quasi-Newton step is given by the updates f_c which minimize the Taylor approximation. Equation (5.39b) shows that each individual parameter of that update can be computed independently by solving the weighted least squares problem

$$f_c[i] = \arg\min_{f}\sum_{t=1}^{T}\hat{\mathrm{E}}_{(-W_t[c,i])}\Big[\big(f - \frac{\nabla_t[c,i]}{-W_t[c,i]}\big)^2\Big]. \tag{5.40}$$

Each individual parameter can either be an element $f_c[j,k]$ of the model matrix of a pairwise factor template or a model function $f_{c,j}(\mathbf{x})$ of a local factor template. The following subsections will go into each of these cases, determine the derivatives, and show how the least squares problems can be solved.

Note that equation (5.39b) doesn't take into account that some of the diagonal elements of the Hessians W_t may be zero and so the term would be undefined. This happens for the parameters $f_c[i]$ which are not tied to any of the neighboring factors of the variable node Y_t, and so the label probability is not functions of that parameter. Consequently, entire summands in the first line (5.39a) are zero and those as well as the respective summands in (5.39b) and (5.39a) should be dropped. The formally correct formulas are deferred until the pseudo-code listing of virtual evidence boosting (algorithm 5.1).

5.4.3 (ii) Newton Steps for Local Factors

The quasi-Newton updates for the local factors are virtually identical to the LogitBoost update steps. The derivatives of the label probabilities $\mathrm{P}(Y_t \,|\, \mathbf{X})$ with respect to the model functions $f_{c,j}(\mathbf{x}_j)$ are

$$\nabla_t(y_t^*, \mathbf{x})[c,j] = \frac{\partial \log \mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})}{\partial f_{c,j}(\mathbf{x}_j)}\Big|_{f_{\langle\Psi\rangle}\equiv 0} \tag{5.41}$$

$$= \mathbf{1}_{\langle y_t^* = j\rangle} - \mathrm{P}(Y_t = j \,|\, \mathbf{x}; F_{\langle\Psi\rangle})$$

and

$$W_t(y_t^*, \mathbf{x})[c,j] = \frac{\partial^2 \log \mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})}{\partial^2 f_{c,j}(\mathbf{x}_t)}\Big|_{f_{\langle\Psi\rangle}\equiv 0} \qquad (5.42)$$
$$= \mathrm{P}(Y_j = j \,|\, \mathbf{x})\big(1 - \mathrm{P}(Y_t = j \,|\, \mathbf{x})\big)$$

if the local factor $\psi_{\{Y_t,\mathbf{X}_t\}}$ of the node Y_t belongs to the template Ψ_c. Otherwise both derivatives are zero. The weighted least squares problems are equivalent to a weighted regressions of each of the update functions $f_{c,j}$ to the points $\big(\mathbf{x}_t, \frac{\nabla_t(y_t^*,\mathbf{x})[c,j]}{W_t(y_t^*,\mathbf{x})[c,j]}\big)$ with weights $(-W_t)$. From equation (5.40) it can be concluded that there is a target point for every training sample and each random variable Y_t connected to a local factor of type c. Since in practice the number of trainings samples can be assumed to one due to parameter tying (cf. section 5.2), there is (at most) one target point for each label variable. The symbol y_t^* stands for the true label of Y_t according to that one training sample, i. e.

$$y_t^* \stackrel{\mathrm{def}}{=} y_t^{(s)}.$$

Any function approximator may be used to fit these points. The experiments in this thesis (section 6.3) fit step functions

$$f_{step}(\mathbf{x}) = \begin{cases} \alpha_1 & \text{if } \pi_i(\mathbf{x}) < \beta, \\ \alpha_2 & \text{if } \pi_i(\mathbf{x}) \geq \beta, \end{cases} \qquad (5.43)$$

if the feature values $\mathbf{x}$ are continuous, or arbitrary functions $f_{B,j} : \mathrm{dom}(\mathbf{X}_t) \mapsto \mathbb{R}$ if the feature values are discrete. The latter simply results in a weighted mean for each element of $\mathrm{dom}(\mathbf{X}_t)$.

For the step function there are four internal parameters to be determined. The first one is the index i which selects one of the components of the features values. ($\pi_i(\mathbf{x})$ is the projection on the i-th entry of the vector $\mathbf{x}$.) In order to pick the best feature, the remaining parameters are determined for each i and then the step function is chosen that minimizes the weighted squared error from equation (5.40). The threshold does not have to be searched but can be determined by a heuristic to maximize the information gain [Liao, 2006]:

$$\beta = \frac{\sum_{t=1}^{T}(-W_t)\, x_t}{\sum_{t=1}^{T} -W_t}. \qquad (5.44)$$

Finally the function values α_1 and α_2 are the weighted mean of the targets on either side of the threshold.

5.4.3 (iii) Stable Newton Steps for Pairwise Factors

For the parameters of pairwise factors computing the Newton steps poses some more challenges. First of all, the derivatives can get very complicated, especially if a hidden node Y_t is neighbor of multiple factors with the same parameters. Secondly, the Taylor approximation of the update criterion may be convex (which is the bad case, as the likelihood is maximized), and so a heuristic has to be found to still produce reasonably scaled and stable updates.

So to avoid the complicated derivatives for a start, we'll go back to the simple CRF from figure figure 5.8 on page 54 and set up as well as solve the least squares problems for the only pairwise factor ψ_A. The general case will follow in the next subsection 5.4.3 (iv).

Example 5.4. Recall that the label probability for the variable Y_1 from the update criterion is

$$\begin{aligned}&\mathrm{P}(Y_1 = y_1^* \,|\, \mathbf{x}; F_A + f_A, F_B + f_B, F_C) = \\ &\qquad \frac{\sum_{y_2} \exp\{F_B(y_1^*, \mathbf{x}_1) + f_B(y_1^*, \mathbf{x}_1) + F_A[y_1^*, y_2] + f_A[y_1^*, y_2] + n_{2\to A}[y_2]\}}{\sum_{y_1, y_2} \exp\{F_B(y_1, \mathbf{x}_1) + f_B(y_1, \mathbf{x}_1) + F_A[y_1, y_2] + f_A[y_1, y_2] + n_{2\to A}[y_2]\}}.\end{aligned} \tag{5.45}$$

With some standard calculus it is easy to see that the derivatives w. r. t. the model matrix component $f[j,k]$ of the logarithm of that term are

$$\begin{aligned}\nabla_1(y_1^*, \mathbf{x})[A, j, k] &= \frac{\partial \log \mathrm{P}(Y_1 = y_1^* \,|\, \mathbf{x}; F_A + f_A, F_B + f_B, F_C)}{\partial f_A[j,k]}\Big|_{f_A \equiv 0, f_B \equiv 0} \\ &= \mathbf{1}_{\langle y_1^* = j \rangle} \frac{\exp\{F_B(j, \mathbf{x}_1) + F_A[j,k] + n_{2\to A}[k]\}}{\sum_{y_2} \exp\{F_B(j, \mathbf{x}_1) + F_A[j, y_2] + n_{2\to A}[y_2]\}} \\ &\quad - \frac{\exp\{F_B(j, \mathbf{x}_1) + F_A[j,k] + n_{2\to A}[k]\}}{\sum_{y_1, y_2} \exp\{F_B(y_1, \mathbf{x}_1) + F_A[y_1, y_2] + n_{2\to A}[y_2]\}} \\ &= \mathbf{1}_{\langle y_1^* = j \rangle} \mathrm{P}(Y_2 = k \,|\, Y_1 = j, \mathbf{x}) - \mathrm{P}(Y_1 = j, Y_2 = k \,|\, \mathbf{x})\end{aligned} \tag{5.46}$$

and

$$\begin{aligned}W_1(y_1^*, \mathbf{x})[i_{j,k}] = \mathbf{1}_{\langle y_1^* = j \rangle} &\mathrm{P}(Y_2 = k \,|\, Y_1 = j, \mathbf{x})\big(1 - \mathrm{P}(Y_2 = k \,|\, Y_1 = j, \mathbf{x})\big) \\ &- \mathrm{P}(Y_1 = j, Y_2 = k \,|\, \mathbf{x})\big(1 - \mathrm{P}(Y_1 = j, Y_2 = k \,|\, \mathbf{x})\big)\end{aligned} \tag{5.47}$$

After differentiating the updates are set to zero because the Taylor approximation only needs the derivatives "evaluated" at the working point, which is $f_A \equiv 0$, $f_B \equiv 0$, and $f_C \equiv 0$. The indicator term in (5.46) results from the fact that the numerator of (5.45) may not contain the parameter $f_A[j,k]$, namely if the true label y_1^* is not equal to j, and

hence the log of the numerator differentiates to zero. Note that unlike in the case of local factors (cf. equation 5.42), the indicator term doesn't vanish in the second derivative.

The derivatives of the second label probability $\mathrm{P}(Y_2 = y_2^* \,|\, \mathbf{x}; F_A + f_A, F_B, F_C + f_C)$ with respect to the same parameter $f_A[j,k]$ yields the equivalent with the roles of Y_1 and Y_2 and of j and k switched. It should be clear that y_2^* has to be equal to k for the indicator term to be non-zero, because Y_2 is connected to the "other end" of the factor ψ_A.

The weighted least squares problem for the scalar parameter $f_A[j,k]$ can be simply solved by differentiating (5.40) w. r. t. that parameter and by setting the result to zero. The solution and hence the quasi-Newton update for this parameter is the weighted mean of the fractions $\nabla_t/(-W_t)$ with weights $(-W_t)$, or equivalently with a simplified numerator:

$$f_A[j,k] = \frac{\sum_{t=1}^{2} \nabla_t(y_j^*, \mathbf{x}_j)[A,j,k]}{\sum_{t=1}^{2} -W_t(y_j^*, \mathbf{x}_j)[A,j,k]} \tag{5.48}$$

This small example already reveals a problem with the Newton step for the pairwise factors: Since there may be a positive summand in the second derivative W_t (5.47), the weights $(-W_t)$ in the weighted least squares problem may be negative. This is a quite unintuitive case, so it is better to go back one step in the derivation and investigate the Taylor polynomial (5.39a). For each individual parameter $f_c[i]$ it contains the following addend:

$$-\frac{1}{2}\,\hat{\mathrm{E}}\big(\sum_{t=1}^{T} -W_t[c,i]\, f_c[i]^2 - 2\nabla_t[c,i]\, f_c[i]\big). \tag{5.49}$$

This is a function of $f_c[i]$ and its graph is a parabola which opens down if and only if the sum of the weights $\hat{\mathrm{E}}\big(\sum_{t=1}^{T} -W_t[c,i]\big)$ is positive. In that case the best update $f_c[i]$ with respect to (5.49) is the vertex of the parabola because we are aiming to maximize the update criterion. Should however the sum of the weights be negative, the parabola opens up and so according to the approximation the best likelihood can be achieved with the current parameter at both positive and negative infinity. This is obviously not true and so in that case Newton's method doesn't result in any useful update steps at all.

Even if the sum of the weights is positive, it can be observed in experiments that the resulting updates are often very extreme and quickly make the algorithm diverge. This can be prevented by imposing some kind of lower bound on the weights, since the vertex computes analogously to equation (5.48). Instead of limiting the sum of the weights, it has been found to work better to limit each weight individually. The Taylor polynomial

of a single label probability with respect to a single factor is the parabola

$$-(-W_t[c,i])\, f_c[i]^2 + 2\nabla_t[c,i]\, f_c[i] \tag{5.50}$$

with the vertex at the abscissa $f_c[i] = \frac{\nabla_t[c,i]}{-W_t[c,i]}$. If we assume that the label probability takes its maximum at a parameter which is close to the current one, for example less than ϑ away, this constraint $|f_c[i]| \leq \vartheta$ should be inserted into the approximation that is used to compute the update. It is easy to see that this can be achieved by setting the weights to

$$(-W_t[c,i]) \leftarrow \max\{(-W_t[c,i]), \frac{\vartheta}{|\nabla_t[c,i]|}\}. \tag{5.51}$$

With this approach the update $f_c[i]$ is guaranteed to not exceed the step limit ϑ in magnitude. Also, the limit is rarely even reached since the clipping is done before taking the weighted average. Experimentally, a step limit of $\vartheta = 2$ has been found successful, but other parameters or entirely different stabilization approaches may be explored in future work.

The first publication of VEB [Liao et al., 2007, Liao, 2006] does not update the pairwise factors by Newton's method, but just solves some other weighted least squares problem whose weights are always positive. Their approach is discussed in the later section 5.4.6.

5.4.3 (iv) Updating Pairwise Factors with Tied Parameters

In many cases tied parameters in the CRF simply results in more summands in the weighted least squares problem (5.40) for the template parameter. This is for example true for local factors, where every hidden node which has a local factor from a template type contributes a term. However the situation is more complicated if one hidden node is connected to more than one factor from a template Ψ_c. Since these factors are updated with the same parameters $f_c[i]$, less summands cancel out in the derivatives of the label probabilities. In this section we will develop a specialized notation for the label probabilities and differentiate that expression. At the end of the section a computationally less complex approximation to the derivatives will be presented.

The general formula for the label probabilities has been presented before as equation (5.31) on page 54. In order to write down the derivatives of that formula, we need to distinguish which templates each of the neighboring factors belong to. Therefore a specialized version of (5.31) needs to be developed. In this context we observe that if we are taking the derivative with respect to $f_c[j,k]$, we can ignore the parameter update for all other factor templates. (Or by example: in (5.46) the derivative would have been the

same if the factor ψ_B had the model functions F_B instead of $F_B + f_B$.) Therefore we can resort to a label probability formula which only makes the parameter F_c explicit. This is achieved by starting from the product of all incoming messages at the variable node (the marginal probability according to belief propagation, cf. equation (5.18)) and only unroll the messages from those factors that have the template type Ψ_c:

$$\begin{aligned}\mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}) \propto \\ \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\Big\{ \sum_{Y_s \in \Upsilon_l} F_c[y_s, y_t^*] + n_{s \to \{Y_s, Y_t\}}[y_s] + \sum_{Y_s \in \Upsilon_l} F_c[y_t^*, y_s] + n_{s \to \{Y_s, Y_t\}}[y_s] \Big\} \\ \times \prod_{Y_{t'} \in \mathrm{nbv}(Y_t) \setminus (\Upsilon_l \cup \Upsilon_r)} m_{\{Y_{t'}, Y_t\} \to t}[y_t^*]. \end{aligned} \tag{5.52}$$

The sets $\mathrm{nbv}(Y_t)$ (cf. equation 5.32), Υ_l, and Υ_r comprise all or a subset of the label variables which are connected to Y_t's pairwise factors. The latter two are defined as

$$\Upsilon_l = \{Y_s \,|\, \exists \psi_A \in \Psi_c, A = (Y_s, Y_t)\} \tag{5.53}$$

$$\Upsilon_r = \{Y_s \,|\, \exists \psi_A \in \Psi_c, A = (Y_t, Y_s)\} \tag{5.54}$$

and stand for the variables which are "to the left" and "to the right" of Y_t and connected by a factor from template Ψ_c. The ends of the pairwise factors need to be distinguished in case both arguments of the factors have the same domain, and so Y_t can be connected to either end.[6]

Replacing the model matrix F_c by $F_c + f_c$ in (5.52) yields the label probability formula which can be derived with respect to the update components $f_c[j, k]$. Note that the normalization, one over the sum ranging over all labels y_t of the entire term, needs to be explicitly taken into consideration for the derivatives. A major difference to the differentiation in example 5.4 in the preceding subsection is that the chain rule yields factors greater than one. Let $\mathtt{ch}(y_t^*, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k)$ denote the number of times a parameter $F_c[j, k]$ occurs in the exponent in (5.52), i. e.

$$\mathtt{ch}(y_t^*, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k) = \sum_{Y_s \in \Upsilon_l} \mathbf{1}_{\langle y_s = j \wedge y_t^* = k \rangle} + \sum_{Y_s \in \Upsilon_l} \mathbf{1}_{\langle y_t^* = j \wedge y_s = k \rangle} \,. \tag{5.55}$$

Also let $\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})$ abbreviate the exponent in equation (5.52) and $\mathtt{mp}(y_t)$ the product of the messages from other factors, each with y_t^* substituted by y_t. The derivatives

[6] This is standard in temporal models for the factors connecting the state variables in subsequent time slices (cf. figure 6.9 on page 94). It is in fact be possible to define these factors to be symmetric, which can be achieved by having both sums in (5.55) range over $\Upsilon_l \cup \Upsilon_r$.

for the Taylor polynomial can then be written as

$$\begin{aligned}\nabla_t(y_t^*, \mathbf{x})[c,j,k] = &\frac{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t^*, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k) \cdot \mathtt{mp}(y_t^*)}{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t^*)} \\ &- \frac{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k) \cdot \mathtt{mp}(y_t)}{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t)}\end{aligned} \tag{5.56}$$

and

$$\begin{aligned}W_t(y_t^*, \mathbf{x})[c,j,k] = &\frac{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t^*, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k)^2 \cdot \mathtt{mp}(y_t^*)}{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t^*)} \\ &- \left(\frac{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t^*, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k) \cdot \mathtt{mp}(y_t^*)}{\sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t^*, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t^*)} \right)^2 \\ &- \frac{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k)^2 \cdot \mathtt{mp}(y_t)}{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t)} \\ &+ \left(\frac{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{ch}(y_t, y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}, j, k) \cdot \mathtt{mp}(y_t)}{\sum_{y_t} \sum_{\mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle}} \exp\{\mathtt{pot}(y_t, \mathbf{y}_{\langle \Upsilon_l \cup \Upsilon_r \rangle})\} \cdot \mathtt{mp}(y_t)} \right)^2\end{aligned} \tag{5.57}$$

Computing these gradients is always feasible, but the learning process is notably slowed down if there are more than two neighboring factors from a single template. This is because the sum over the neighbor assignments $y_{\langle \Upsilon_l \cup \Upsilon_r \rangle}$ has $\mathrm{dom}(Y_s)$ to the power $|\Upsilon_l \cup \Upsilon_r|$ summands (Y_s is any element of $\Upsilon_l \cup \Upsilon_r$). Furthermore, this runtime is required once for every hidden node in the CRF and every factor template the node's neighbors belong to (but not for the elements of the model matrices). An efficient implementation requires only one depth-first search over all assignments which at the same time accumulates the potentials and counts parameter occurrences (one summand per recursion) and sums over the assignments (one summand every time a leaf is reached). Using an amount of space which is linear in the number of model matrix parameters, this allows to compute the derivatives with respect to all model matrix parameters $f_c[j,k]$, $j = 1 \ldots J$, $k = 1 \ldots K$. With these derivatives the stabilized Newton steps are computed just as described in the preceding subsection.

At this point the reader may ask why the computations for local factors don't take exponential runtime in the number of neighboring factors. The basis of the derivatives are the label likelihoods (5.31) which do sum over all possible label combinations for the Markov blanket. In the previous sections however all sums could be collapsed back into belief propagation messages after taking the derivatives and setting all parameter updates

$f_{\langle\Psi\rangle}$ to zero. Equivalently, this step can also be done before taking the derivatives, yielding label probability formulas like in (5.52) but with $\Upsilon_l \cup \Upsilon_r$ containing at most one label variable. For taking the derivatives with respect to the parameters of local factors $\psi_{\{Y_t,\mathbf{X}_t\}}$, respectively pairwise factors $\psi_{\{Y_t,Y_s\}}$ with unique parameters, the label probability formulae can be simplified to

$$\mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}) \propto \exp\{F_{\{Y_t,\mathbf{X}_t\}}(y_t^*, \mathbf{x}_t)\} \times \prod_{Y_{t'} \in \mathrm{nbv}(Y_t) \setminus \{Y_s\}} m_{\{Y_{t'},Y_t\}\to t}[y_t^*] \tag{5.58}$$

respectively

$$\mathrm{P}(Y_t = y_t^* \,|\, \mathbf{x}) \propto \sum_{y_s} \exp\{F_{\{Y_t,Y_s\}}[y_t^*, y_s]\} \times \prod_{Y_{t'} \in \mathrm{nbv}(Y_t) \setminus \{Y_s\}} m_{\{Y_{t'},Y_t\}\to t}[y_t^*]. \tag{5.59}$$

The derivatives of these terms have the same complexity as the terms themselves, so computational cost was previously not an issue.

We observe that in each of these formulae only one of the messages arriving at Y_t is unrolled. This idea can also be applied if a hidden node is connected to several factor with tied parameters: instead of unrolling several messages at once like in (5.52), we unroll them separately and take the average. More precisely, this can be described as a modification of the update criterion (5.28) specific to the template currently updated. Each of its summands

$$\log \mathrm{P}(Y_t \,|\, \mathbf{X}; F_{\langle \mathrm{nb}(Y_t)\rangle} + f_{\langle \mathrm{nb}(Y_t)\rangle}, F_{\langle \Psi \setminus \mathrm{nb}(Y_t)\rangle})$$

is replaced by one term for each pairwise factor of the current template. These sum to

$$\frac{1}{|\Upsilon_l \cup \Upsilon_r|} \sum_{Y_s \in \Upsilon_l \cup \Upsilon_r} \log \mathrm{P}(Y_t \,|\, \mathbf{X}; F_{\{Y_t,Y_s\}} + f_{\{Y_t,Y_s\}}, F_{\langle \Psi \setminus \psi_{\{Y_t,Y_s\}}\rangle}) \tag{5.60}$$

The summands take the simple form of (5.59), and so the derivatives of the sum are in analogy to example 5.4 on page 60

$$\nabla_t(y_t^*, \mathbf{x})[c,j,k] = \eta \sum_{Y_s \in \Upsilon_l \cup \Upsilon_r} \mathbf{1}_{\langle y_t^* = j\rangle} \, \mathrm{P}(Y_s = k \,|\, Y_t = j, \mathbf{x}) - \mathrm{P}(Y_t = j, Y_s = k \,|\, \mathbf{x}) \tag{5.61}$$

and

$$\begin{aligned} W_t(y_t^*, \mathbf{x})[c,j,k] = \eta \sum_{Y_s \in \Upsilon_l \cup \Upsilon_r} & \mathbf{1}_{\langle y_t^* = j\rangle} \, \mathrm{P}(Y_s = k \,|\, Y_t = j, \mathbf{x})\big(1 - \mathrm{P}(Y_s = k \,|\, Y_t = j, \mathbf{x})\big) \\ & - \mathrm{P}(Y_t = j, Y_s = k \,|\, \mathbf{x})\big(1 - \mathrm{P}(Y_t = j, Y_s = k \,|\, \mathbf{x})\big) \end{aligned} \tag{5.62}$$

with the factor $\eta = 1/\,|\Upsilon_l \cup \Upsilon_r|$.

We will call this approach for computing the updates of pairwise factors *piecewise Newton steps* (based on the name of related piecewise approximation of pseudo-likelihood training by Sutton and McCallum [2007]) and the more exact version above *neighborhood-based Newton steps*. The resulting versions of VEB are compared empirically in section 6.3.

5.4.4 Selecting the Factor with the Best Update

Many applications of boosting choose to use very simple weak classifiers or function approximators like decision trees with few levels or even decision stumps [e. g. Schapire and Singer, 1999, Friedman et al., 2000]. It has been observed that this yields a good generalization performance which may be caused by one or both of the following points: The family of distribution that can be represented is smaller when simple base functions are used, which may make it harder to overfit. Secondly, it makes the algorithm select the best features. We will now explore the latter aspect and show how this is put into practice in VEB.

The key property of decision stumps and low decision trees is that they are only functions of one or a few feature values x. So when a tree or stump is fitted, features are picked that yield the best update in the current iteration. Consequently, there is a fair chance that all features that have been chosen and integrated into the model are actually significant for the labels. The other way round, if there is a feature that is entirely random, it may still have a correlation with the labels in the training set but this correlation will usually be weaker than correlations of significant features. If this is the case in all iterations, the random feature will be ignored entirely by the model. Therefore feature selection may indeed contribute to a good generalization performance.

In conditional random fields there are not only different features but they may also be connected to distinct hidden variables (via local factors), and there are pairwise "features" between the hidden variables. So to generalize the concept of feature selection to CRFs, VEB only updates one factor (respectively one set of factors with tied parameters) per iteration. This leads to the already described bootstrapping behavior of VEB: In the first iterations the algorithm mostly updates local factors and only later learns the correlation between the labels. Also, it allows VEB to detect relevant dependencies between the hidden variables and ignore irrelevant ones (cf. experiments in [Liao et al., 2007]). With the decision stumps as base functions for the local factors (cf. section 5.4.3 (ii)) VEB also selects the best features whenever a local factor is updated.

In order to determine the factor to be updated, candidate updates need to be computed for every one of them. Then the best update can be simply determined by eval-

uating the non-constant parts of equation (5.39b). The improvement of the per-label likelihood by an update f_c is approximately

$$\frac{1}{2}\sum_i \sum_{t=1}^{T} \Big(\hat{\mathrm{E}}\Big[\frac{\nabla_t[c,i]^2}{-W_t[c,i]}\Big] - \hat{\mathrm{E}}_{(-W_t[c,i])}\Big[\big(f_c[i] - \frac{\nabla_t[c,i]}{-W_t[c,i]}\big)^2\Big]\Big), \tag{5.63}$$

where i ranges over all parameters of the factor template Ψ_c. (Again all summands with $W_t[c,i] = 0$ should be omitted; cf. section 5.4.3 (i).) The first term can be interpreted as the maximum improvement of the per-label likelihood for a factor, and by deducing the fitting error we get the actual improvement. To summarize, only the additive update for the factor template Ψ_c maximizing (5.63) is applied (see also an alternative in the following section) and that concludes an iteration of VEB.

Apart from the reasons given above, the factor selection approach can be justified from an optimization viewpoint. The quasi-Newton steps assume that each parameter affects the optimization criterion in an uncorrelated manner. This is clearly an approximation, and so if the parameters have correlated effects, the quasi-Newton step may be too large. Quasi-Newton and full Newton stepping are only equivalent if only a single scalar parameter is optimized. Therefore quasi-Newton steps may be more accurate for lower numbers of parameters. This is one side-effect of factor selection and hence may contribute the algorithm's stability.

5.4.5 Practical Considerations and Implementation

The training method virtual evidence boosting in the versions *VEB with neighborhood-based Newton steps* and *VEB with piecewise newton steps* is listed in algorithm 5.1 on the following pages.

Many practical consideration have already been discussed in the derivation in the previous sections. However the actual implementation contains two notable modifications of the vanilla algorithm to ensure the usability for a wide variety of applications.

Since the optimization process of VEB may diverge once a model with strong label dependencies has been learned (see section 5.4.2), this case needs to be handled gracefully. An approach which has been found to work well in practice is to monitor the classification performance on the training set and keep a backup of all parameters from the iteration that scored best on that measure. This comes at no significant extra cost: the marginal probabilities are trivially determined from the belief propagation messages and we only base the decision for the best labels on marginal probabilities (cf. section 5.3).

Input: a CRF with the factor templates Ψ_c; the training data, an instantiation $(\mathbf{x}, \mathbf{y}^*)$ of all random variables; a function approximator; the number of learning iterations M

Output: the trained CRF, i. e. the factor function ψ_c for each template Ψ_c

intialize model functions $F_{c,j} \equiv 0$ and model matrices $F_c[j,k] = 0$

define the functions of local and pairwise factors

$$\psi_c(y, \mathbf{x}) = \exp\{\sum_{j=1}^{J} \mathbf{1}_{\langle y=j \rangle} F_{c,j}(\mathbf{x})\} \quad \text{respectively} \quad \psi_c(y, y') = \exp\{F[y, y']\}$$

for $m = 1$ **to** M **do**

  run loopy belief propagation on the CRF

  foreach factor template index $c \in \{1, \ldots, C\}$ **do**

    if Ψ_c is a pairwise factor template **then**

      $(f_c, g_c) =$ `pairwise_factor_step(`c`)`

    else

      $(f_c, g_c) =$ `local_factor_step(`c`)`

    end

  end

  select the factor template yielding the best improvement $c^* = \arg\max_c g_c$

  $F_{c^*} \leftarrow$ `apply_update(`c^*`,`F_{c^*}`,`f_{c^*}`)`

end

Algorithm 5.1: Virtual evidence boosting for training conditional random fields.

Function: `local_factor_step`

Input: index c of a local factor template; the current CRF; training data $(\mathbf{x}, \mathbf{y}^*)$; belief propagation messages from last run; a function approximator for weighted least squares fitting

Output: the update f_c to the model functions of template Ψ_c and its likelihood gain g_c

initialize $g_c = 0$
foreach model function $F_{c,j}$ **do**
  foreach node index $t \in \{1, \ldots, T\}$ **do**
    if the node Y_t has a local factor $\psi_B \in \Psi_c$ **then**
      compute the marginal probability for class j

$$p = \frac{\prod_{\psi_A \in \mathrm{nb}(Y_t)} m_{A \to t}[j]}{\sum_k \prod_{\psi_A \in \mathrm{nb}(Y_t)} m_{A \to t}[k]}$$

      compute weight and working response numerator

$$w_t = p(1-p); \qquad d_t = \mathbf{1}_{\langle y_t^* = j \rangle} - p$$

    else
      set $d_t = 0$, $w_t = 0$
    end
  end
  use the function approximator to fit the function $f_{c,j}(\mathbf{x})$ to the points $\{(\mathbf{x}_t, \frac{d_t}{w_t}) \,|\, d_t \neq 0\}$ by weighted least squares with weights w_t
  accumulate the likelihood gain $g_c \leftarrow g_c + \sum_{t, d_t \neq 0} \frac{d_t^2}{w_t} - w_t \cdot \left(f_{c,j}(\mathbf{x}_t) - \frac{d_t}{w_t}\right)^2$
end

Function `local_factor_step` Finds best base function approximating a quasi-Newton step for local factor functions.

Function: `pairwise_factor_step`
Input: index c of a pairwise factor template; the current CRF; training data $(\mathbf{x}, \mathbf{y}^*)$; belief propagation messages from last run
Output: the model matrix update f_c and its likelihood gain g_c

initialize $g_c = 0$
foreach model matrix parameter $F_c[j,k]$ **do**
  foreach node index $t \in \{1, \dots, T\}$ **do**
    compute the derivatives (5.56)–(5.57) or (5.61)–(5.62) and set

$$d_t = \nabla_t(y_t^*, \mathbf{x})[c,j,k]$$
$$w_t = -W_t(y_t^*, \mathbf{x})[c,j,k]$$

    if $d_t \neq 0$ **then** apply step limit stabilization $w_t \leftarrow \max\{w_t, \frac{2}{|d_t|}\}$
  end
  compute the parameter update $f_c[j,k] = \big(\sum_{t=1}^{T} d_t\big) / \big(\sum_{t=1}^{T} w_t\big)$
  accumulate the likelihood gain $g_c \leftarrow g_c + \sum_{t,d_t \neq 0} \frac{d_t^2}{w_t} - w_t \cdot \big(f_c[j,k] - \frac{d_t}{w_t}\big)^2$
end

Function `pairwise_factor_step` Stabilized quasi-Newton step for pairwise factors.

Function: `apply_update`
Input: a factor template index c; the current model parameters F_c for that template; the update f_c
Output: the updated model parameters F_c

if Ψ_c is a local factor template **then**
  foreach model function $F_{c,j}$ **do**
    center and discount the update $f'_{c,j} = \frac{J-1}{J}\big(f_{c,j} - \frac{1}{J}\sum_{j'=1}^{J} f_{c,j'}\big)$
    add the update $F_{c,j} \leftarrow F_{c,j} + f'_{c,j}$
  end
else
  foreach model matrix component $F_c[j,k]$ **do**
    center and discount $f'_c[j,k] = \frac{JK-1}{JK}\big(f'_c[j,k] - \frac{1}{JK}\sum_{j'=1}^{J}\sum_{k'=1}^{K} f_c[j',k']\big)$
    apply $F_c[j,k] \leftarrow F_c[j,k] + f'_c[j,k]$
  end
end

Function `apply_update` Applies the update taking the parameterization restrictions into account (see section 5.4.1)

The second modification concerns the factor selection strategy introduced in the preceding section. Updating only one factor per iteration may slow down the learning if there are many different factor templates. Furthermore if the number of factors per template have substantial differences, the ones with low counts may never be touched. So to meet both these problems, multiple templates are updated so that for every "label variable type" (the sets of variables which have identical domains and the same semantic interpretation) this involves at least one factor template which connects to label nodes of that type. The only exception to this rule is for label variable types which don't have a local factor because during the bootstrapping phase there may not yet be any pairwise label correlations to learn. This ensures that the factors are updated more evenly and quickly, while still applying the principle of feature selection.

5.4.6 Comparison with Liao's Virtual Evidence Boosting

As aforementioned the ideas and principles of virtual evidence boosting have been published by Liao et al. [2007]. They claim to generalize LogitBoost for training conditional random fields, but their updates to pairwise factors are not equivalent to Newton steps, the optimization method of LogitBoost. In this section we point out some details about their derivation and compare the differences between Liao's VEB and our versions of VEB with Newton steps.

The difference between the methods can indeed be traced back to an error in the derivation of Liao's VEB, which was published in [Liao, 2006]. The methods still agree on the criterion used to compute the update iterations: both optimize the log label probabilities summed over all hidden nodes with each label probability term equivalent to (5.31). In the following Newton step computations however, Liao does not distinguish the differentiation with respect to the model functions of local factors and the model matrix elements of pairwise factors. He differentiates the label probabilities (5.31) with respect to a column $f_c[j, \cdot]$ of the model matrix, but fails to see that the column elements are different parameters. Instead, he implicitly uses

$$\frac{\partial f_c[j,k']}{\partial f_c[j,k]} \stackrel{!}{=} 1 \quad \text{for all } k' \text{ in its domain } \{1,\ldots,K\},\ K \geq 2,$$

which is obviously false. This makes the resulting derivatives independent of the neighboring label variables, and that dependency is reintroduced by taking an unjustified additional weighted expectation over the neighbor labels in the weighted least squares problem. This step is incorrect because despite their notation "$P(y_t \,|\, \mathbf{ve}(\mathbf{x}_t))$",[7] the

[7] The vector $\mathbf{x}_t$ includes the labels $\mathbf{y}_{\langle \mathrm{nbv}(Y_t) \rangle}$ of the neighboring variable nodes.

formula with respect which the derivatives are taken does not stand for some conditional distribution $\mathrm{P}(Y_t \mid \mathrm{nbv}(Y_t), \mathbf{X})$, which may need to be multiplied by something like $\mathrm{P}(\mathrm{nbv}(Y_t) \mid \mathbf{X})$, but is by their definitions eventually equivalent to the unrolled marginal probability of the CRF (5.31).

Interestingly, we will see in chapter 6.3 that Liao's VEB performs better than VEB with Newton steps. Irrespective of the derivation the factor updates of Liao's VEB have similarities with one of our versions, namely VEB with piecewise Newton steps (cf section 5.4.3 (iv)). Their updates are defined as

$$f_c[j,k] = \frac{\sum_{\psi_{\{Y_t,Y_s\}} \in \Psi_c} (\mathbf{1}_{\langle y_t^* = j \rangle} - \mathrm{P}(Y_t = j \mid \mathbf{x})) \cdot m_{s \to \{Y_t,Y_s\}}[k]}{\sum_{\psi_{\{Y_t,Y_s\}} \in \Psi_c} \mathrm{P}(Y_t = j \mid \mathbf{x})(1 - \mathrm{P}(Y_t = j \mid \mathbf{x})) \cdot m_{s \to \{Y_t,Y_s\}}[k]} \tag{5.64}$$

while the Newton step is

$$f_c[j,k] = \frac{\sum_{\psi_{\{Y_t,Y_s\}} \in \Psi_c} (\mathbf{1}_{\langle y_t^* = j \rangle} - \mathrm{P}(Y_t = j \mid \mathbf{x})) \cdot \mathrm{P}(Y_s = k \mid Y_t = j, \mathbf{x})}{\sum_{\psi_{\{Y_t,Y_s\}} \in \Psi_c} -\mathbf{1}_{\langle y_t^* = j \rangle} [\mathrm{P}(Y_s = k \mid Y_t = j, \mathbf{x})]^* + [\mathrm{P}(Y_j = j, Y_s = k \mid \mathbf{x})]^*} \tag{5.65}$$

where $[p]^*$ abbreviates $p\,(1-p)$.[8] It is easy to see that the numerators in both formulae bear a striking similarity. Taking into account the definition

$$\mathrm{P}(Y_s = k \mid Y_t = j, \mathbf{x}) \propto \exp\{F_{\{Y_t,Y_s\}}[j,k]\} \cdot m_{s \to \{Y_t,Y_s\}}[k], \tag{5.66}$$

they only differ in a single factor. Still, the relationship between the denominators remains unclear. It has to be pointed out however that the Newton steps had to be limited to stabilize the optimization procedure (see section 5.4.3 (iii)), which was done by modifying the denominator of (5.65). This issue never occurs in Liao's derivation because his weights, which are the summands in the denominator of (5.64), are always positive. In fact, it may be possible that — by coincidence — Liao [2006] has found a stabilization heuristic which works particularly well in practice.

An only apparent difference is the criterion for selecting the factor whose update yields the best improvement (cf. section 5.4.4). The improvement quantity (5.63) is the difference of the maximum improvement and the fitting error. Liao et al. [2007] omit the former term, which does not depend on the update f but is usually specific to the variable node Y_t and the factor template to be evaluated. However with their mistake in the computation of the derivatives, the dependency on the factor template

[8] These formulae are simplified for better readability. Every summand should in fact be duplicated with the roles of j and k as well as Y_t and Y_s switched (cf. example 5.4 on page 60). Also the factors η from (5.61)–(5.62), which normalize the contribution of each label to the update criterion are omitted in (5.65). The factors η are not relevant in many CRF topologies.

is eliminated (where their derivatives $\nabla_t(y^*, \mathbf{x})[c, i]$ and $W_t(y^*, \mathbf{x})[c, i]$ are not zero, they are a constant for all c and i). Furthermore the strategy of updating one neighboring factor of every "node type" (see section 5.4.5) causes that those likelihood gains which need to be compared sum over the same sets of hidden nodes Y_t. Therefore omitting the maximum improvement term in Liao's VEB does not affect the selection criterion. The versions agree on the principle, but the actually chosen factors naturally still differ due to the different derivatives.

As a mean to make virtual evidence boosting computationally more efficient, Liao et al. [2007] claim that it is sufficient to run only one iteration of loopy belief propagation at the beginning of each VEB iteration (cf. algorithm 5.1, line 4). Apart from the lacking theoretic justification, we have found that the reported performance during training may be significantly different to the test performance on the training data (!). With the limited inference during training, the algorithm reports a correct classification rate of 99% while the exact inference (the CRF was a linear chain in that setup) on the same data yielded only 93% correct classification. Therefore we decided to always run belief propagation until convergence in all versions of VEB for the following experiments. The iterations of VEB have a comparable runtime to other learning methods requiring inference, but by performing an approximation to Newton steps the number of iterations is much smaller that in other approaches.

Chapter 6

Recognizing Spatial Context

This chapter describes the setup, execution, and results of the activity recognition task pursued in this work. We recognize spatial context from data recorded with a wearable sensor system. The main emphasis is placed on comparing the recognition performance with different sensors and features. Furthermore the experiments are repeated for each of three versions of virtual evidence boosting for an empirical comparison of the algorithms. The setup and features extracted from the two sensor systems, a camera and a multi-sensor integrated device are introduced in the first section. Then we give an overview over the recorded data traces, followed by the experimental results.

6.1 Sensors and Features

The spatial context, the type of place a person is currently in, can be determined by simply measuring properties of the environment. Some of these properties are noise, ambient light, temperature, or barometric pressure, and there are simple sensors to measure them. For humans however the main source of information are the eyes. Still, the corresponding artificial sensor, a digital video camera, has rarely been used in activity recognition applications (cf. related work in section 2.3). This is because there are several issues that are specific to that sensor. First of all, cameras are directed sensors (omnidirectional cameras are hard to accommodate in a wearable setup), so they only show what's right in front of them, which might be quite uninformative. Secondly they produce rich but also very variable data which poses the question of how to design features to extract relevant information. Also the large amount of data may exceed the processing power of a portable device, which is however not a problem in our offline evaluation approach.

Despite these issues, we will investigate the use of camera images for detecting spa-

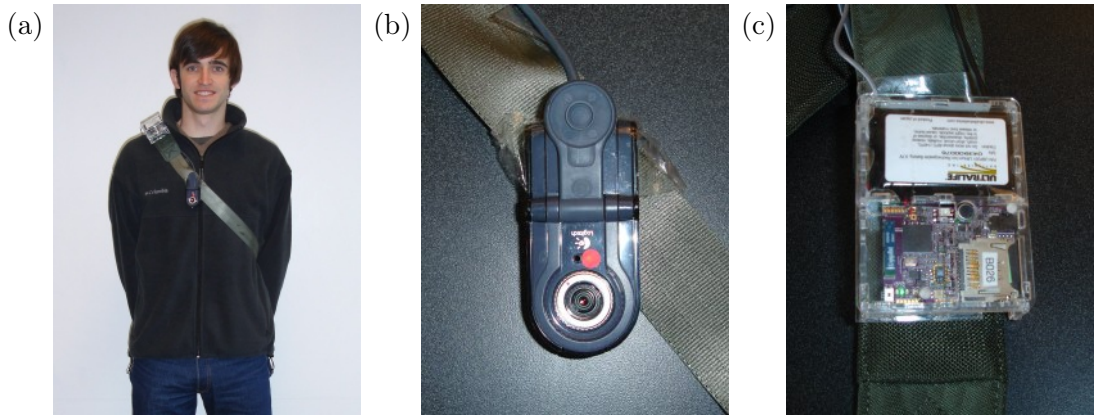

Figure 6.1: Wearable sensor system consisting of a web-cam (b) and the multi-sensor board (c).

tial context and compare them to the results that can be achieved with other sensors, including audio, light, and temperature sensors. To do this, we first need to identify suitable image features. As a baseline, we will include the feature set used by Torralba et al. [2003] to detect individual rooms as well as spatial context categories. A further two feature sets are taken from scene classification research, which attempts to classify photographs by the depicted scene or background (cf. section 2.5). After introducing the hardware in the first subsection, the three image features will be presented in sections two to four, followed by the features extracted from the other sensors modalities.

6.1.1 Sensor Hardware and Setup

The sensor system used for our experiments is very lightweight and puts little burden on the wearer. It includes integrated devices that have been developed at the University of Washington in cooperation with the Intel Research Lab in Seattle. In detail, the components are a Logitech QuickCam For Notebooks Pro, which is connected via USB to an Intel Mote (iMote), and a *multi-sensor board* (MSB) that integrates eight sensor modalities on a tiny footprint of only 48mm × 36mm (cf. figure 6.1c). The MSB is sitting on top of a further iMote, which runs the software for recording and saving the data to a SD Card. The camera system produces a series of 24 bit color images with a resolution of 640×480 pixels at a rate of 1 image per 1.3 seconds.

Both devices are attached to the strap of a shoulder bag, with the MSB clipped on at the shoulder facing upwards and the camera attached at the chest facing forward (see figure 6.1a). The iMote for recording the camera data is placed inside the bag in the back. While this requires the USB cable to run alongside the bag's strap, the camera and both

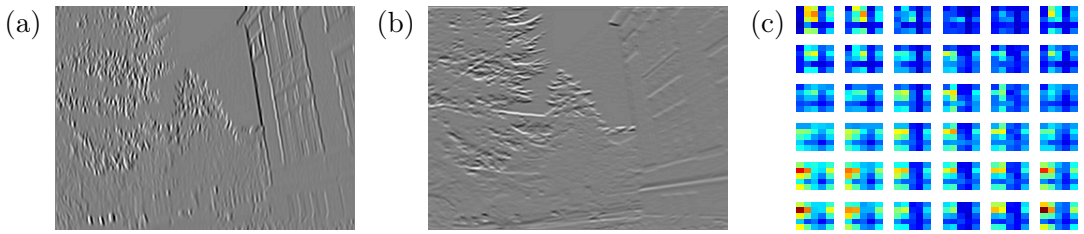

Figure 6.2: (a–b) Response of vertical and horizontal differential filters to an image captured on campus. (c) All steerable pyramid features for that image. Scales are arranged in rows, with the response from small structures at the bottom, and orientations in columns. The first and fourth block in the second row from the bottom are the response magnitudes of (a) and (b).

recording devices could also have been easily integrated in a single case and mounted at the shoulder. The usability of a system strongly depends on the ease of wearing the required devices, and a single device clearly surpasses a system where several devices have to be mounted in different locations of the body [cf. Bao and Intille, 2004, Kern et al., 2003]. However the camera and the light sensors on the MSB always have to be carried visibly which may affect the acceptance of such a system in civil applications.

6.1.2 Steerable Pyramid Differential Filters

A potentially interesting feature is the variability of the light intensity in the image as they result from edges, textures, and objects in the scene. These intensity changes can be characterized by their scale (abrupt or gradual changes) and their orientation (direction of the gradient). A further useful information is the location in the image where changes occur.

In order to capture these properties in a set of feature values, Torralba et al. [2003] use the *steerable pyramid* image decomposition [Simoncelli and Freeman, 1995]. This image processing method decomposes the image into several images that each only contain a certain structure scale and certain structure orientation by applying a set of image filters. For each pixel and each filter this yields a response value indicating the presence of an edge or texture of the respective scale and orientation around the pixel (see figure 6.2 a–b). To extract feature values from the set of filter response images, Torralba et al. [2003] average the magnitudes of each of them over a total of 16 subblocks (4 rows by 4 columns), retaining some information about the location of structures in the image (see figure 6.2c).

The decomposition performed by the steerable pyramid filters is best described in

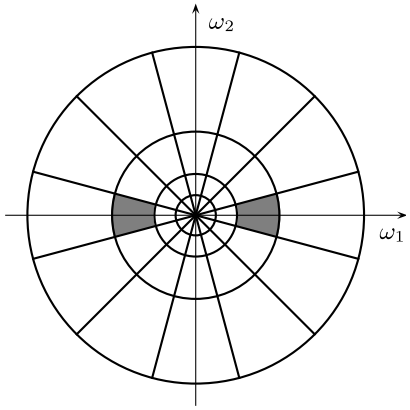

Figure 6.3: Idealized representation of the spectral decomposition performed by the steerable pyramid in Fourier space. The shaded area corresponds to the frequency band passed by the filter used for figure 6.2a. (Figure source: [Simoncelli and Freeman, 1995])

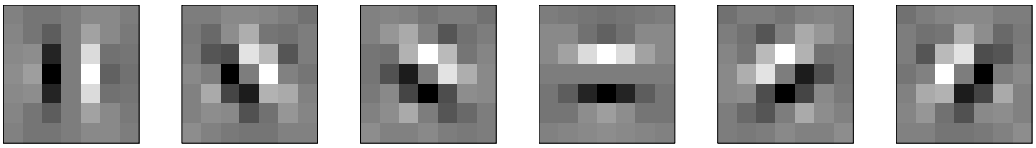

Figure 6.4: Differential filters for the six orientations 0°, 30°, 60°, 90°, 120°, 150°.

the Fourier domain. The scale subbands, rings around the origin, are divided into $2K$ segments (K is the number of orientations), and each of the filters is a band-pass filter for one of these ring segments and its point symmetric opposite (see figure 6.3).

The implementation approximates the ideal filters in the spatial domain in an iterative manner. Similar to a multiscale decomposition like the Gauss-Laplace pyramid, different scales of the image are produced by repeatedly downsampling the image from the preceding iteration to half its resolution. Then, each of the scaled images is convoluted with a set of directional derivative filters that each respond to a distinct angular range (see figure 6.4). These filters are simply rotated ("steered") copies of each other. Since the scaled images have variable sizes and the filters' sizes are fixed (7×7 pixels), this responds to structures at different granularity.

The number of orientations, scales, and averaging windows are parameters that need to be chosen. Following Torralba et al. [2003], six orientations (0°, 30°, 60°, 90°, 120°, 150°) and 16 averaging windows are deemed reasonable. Since the images of Torralba et al. have a quarter of our resolution (theirs is 160×120 pixels), we use six instead of four scales so that the largest ratio of filter to image size is the same.

Instead of using the resulting 384 (here 576) features directly, Torralba et al. [2003] use the first 80 principal components. *Principal component analysis* (PCA) is a well-known dimension reduction method which rotates the feature space so that the variation within the data points occurs in the first dimensions of the rotated space (the "principal components") and hence the remaining dimension can be omitted. This also results in some decorrelation of the input data, which is crucial for generative statistical models which assume independent feature components. While this neither the case here nor in [Torralba et al., 2003] the shorter and hence more concise feature vector — the PCA reduces 576 values to 80 values — may still allow better results. This will be evaluated through experiments (see section 6.3.1).

Motivated by the observation that the location of structures in the image may be very variable, we also evaluate a new version of the steerable pyramid feature set which adds the sums over rows and columns of the image subblocks as features. Recall that the camera is fixed to the body of the wearer, and so the image shifts horizontally when the person turns. With this extension the learning algorithm may for example choose to use the filter response averaged over a horizontal strip of the images.

6.1.3 Histogram Features

Other than grayscale variations, the color distribution in the image should give very strong clues about the spatial context. It has to be noted that the camera performs white balance in an automatic and inaccessible way and hence some color information is lost. However the common color features from scene classification, which work well on automatically balanced photographs, can still be used. This section gives details about the color histogram features used in the experiments, followed by color-spatial moments in the next section.

Definition 6.1. Let an image be defined as an array $\mathtt{im}[i, j]$ of values from a color space $\mathcal{C}$. Let $H = (B_n)_{1 \le n \le N}$ be a partition of the color space $\mathcal{C}$. The histogram $h = (h_n)_{1 \le n \le N}$ with respect to the partition H is then defined as

$$h_n = \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{1}_{\langle \mathtt{im}[i,j] \in B_n \rangle} \ ,$$

i. e. the number of pixels in each of the bins B_n.

In theory the choice of color space is irrelevant for the expressiveness of histograms. In practice however the bins are conveniently chosen to be axially parallel cuboids, which allows to describe them by two inequalities per color channel. Therefore a color space

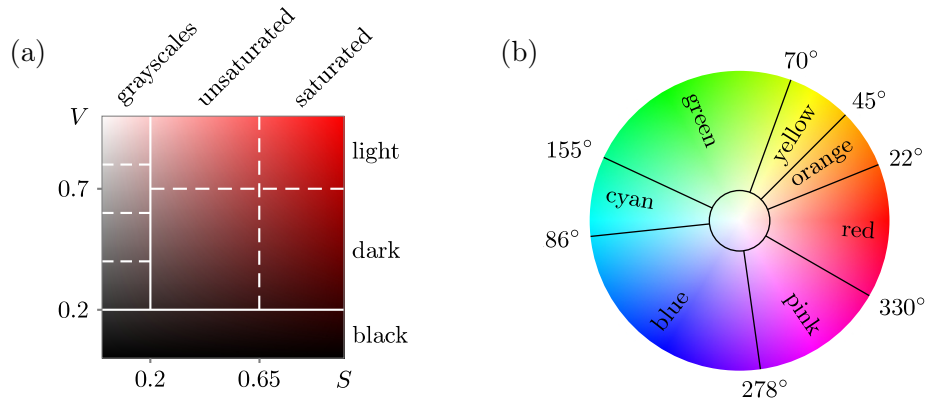

Figure 6.5: (a) Partition by saturation and value. (b) Partition by hue.

should be chosen where the cross-product of channel ranges yields bins which actually contain similar colors. A color space like the *HSV* (hue, saturation, value) space [Smith, 1987] is more suitable for this than other color spaces like *RGB* because the channels capture meaningful concepts which are fairly stable under variations of the other channels. The hue for example is an invariant aspect of color across different brightnesses and saturations, given these are not too low (cf. figure 6.5a). HSV is not perceptually uniform, i. e. the distance between colors in the color space are not proportional to the perceived difference of the colors (even when using the Euclidean distance in the cone-shaped embedding, cf. section 6.1.4). However this can be easily accounted for by using non-uniform bin sizes.

For the bin partition of the HSV color space, we closely follow the approach by Lei et al. [1999]. Regarding the saturation-value plains for fixed hues, three regions are identified (see figure 6.5a): For $V \leq 0.2$ the color is too dark to perceive hue or saturation, so all these colors are put in a bin for black. Colors triples with a very low saturation $S \leq 0.2$ are considered grayscales. We split these into 4 ranges from dark gray to white, yielding a total of 5 hue-independent bins. The remaining color space ($V > 0.2, S > 0.2$) is the range where the hue is taken into account. It is split into color ranges that can be easily named, which approximately yields a perceptually uniform split (figure 6.5b). These colors are red, orange, yellow, green, cyan, blue and pink. Finally, each of the colors is subdivided into four bins: by saturation with a threshold of $S = 0.65$ and by value at $V = 0.7$ (figure 6.5a). The total number of bins is hence $5 + 7 \times 4 = 33$. See figure 6.6c for an example histogram.

This partition has the desirable property of not having too many bins and should not allow for significant overfitting. However it is not clear if the thresholds are chosen well

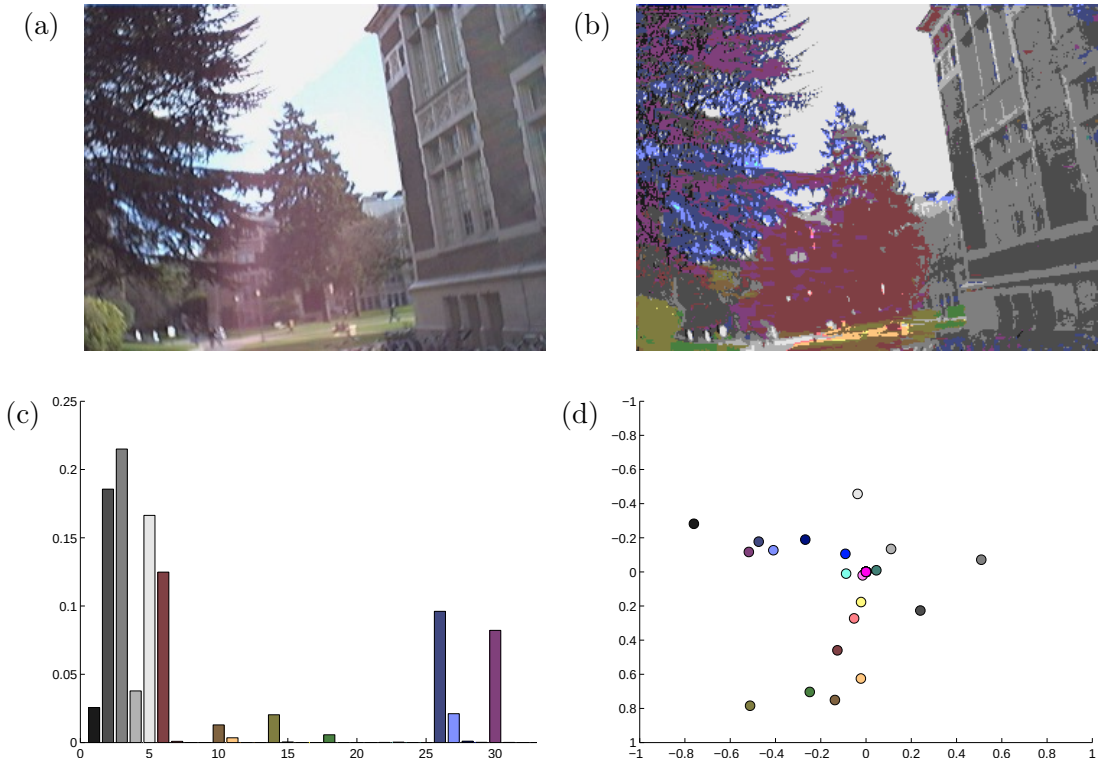

Figure 6.6: (a) Original image. (b) Segmented image. (c) Histogram features. (d) Bin center of mass features.

and so there may a lot of variation across a threshold in similar images. To account for this, histograms with overlapping or fuzzy bins could be used. A simple alternative is to add aggregates of neighboring bins to the feature vector, possibly eliminating the effects of a badly chosen threshold and giving the learning algorithm more yet stable features to choose from. The aggregates added span accross saturations, values, neighboring hues, neighboring grayscales, all grayscales, or all light colors, totaling to 43 new features. This addition will be evaluated empirically in section 6.3.1.

6.1.4 Color-Spatial Moments

Histograms do not capture any information about the spatial distribution, i. e. the position in the image, of colors. This may also contain valuable information about the scene, although, unlike in scene classification tasks [Vailaya et al., 2001, p. 121], our outdoor images usually have a gray rather than blue sky at the top (see figure 6.6 a–b).

While it is thinkable to create a joint histogram over colors and positions, an excessive number of bins would be required to cover the resulting five-dimensional space. So only

some of the dimensions are discretized by a histogram and the marginal distribution over positions or colors are described more concisely by its *moments*, i. e. mean and (co)variance. Both approaches, spatial moments for histogram bins [Lei et al., 1999] and color moments for subblocks of the image [Stricker and Orengo, 1995], are investigated as features for our experiments.

6.1.4 (i) Bin Center of Mass Features

The mean image location $(\bar{\imath}_n, \bar{\jmath}_n)$ of the pixels in histogram bin B_n, commonly referred to as *center of mass*, can be computed as follows:

$$\bar{\imath}_n = \frac{1}{h_n} \sum_{i=1}^{I} \sum_{j=1}^{J} i \cdot \mathbf{1}_{\langle \mathtt{im}[i,j] \in B_n \rangle} \quad \text{and} \quad \bar{\jmath}_n = \frac{1}{h_n} \sum_{i=1}^{I} \sum_{j=1}^{J} j \cdot \mathbf{1}_{\langle \mathtt{im}[i,j] \in B_n \rangle} \ ,$$

with the pixel counts h_n defined as in definition 6.1. Examination of the formula reveals that the center of mass is undefined in case that the bin is empty. Also, it is undesirable that the center of mass may be fairly random in case of few pixels in a bin. An effective way to stabilize the value in both cases is to assume that each bin contains a small, fixed number of pixels at the center of the image. This pulls the the center of mass towards the center of the image as the number of pixels in the bin approaches zero. In a nutshell, the feature values are the stabilized and normalized offsets of the centers of mass to the image center:

$$\bar{\imath}'_n = \frac{1}{h_n + \gamma I J} \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{2i - I}{I} \cdot \mathbf{1}_{\langle \mathtt{im}[i,j] \in B_n \rangle} \ ,$$

and $\bar{\jmath}'_n$ defined accordingly. The stabilization coefficient is chosen to be $\gamma = 0.0025$. With the image resolution of $I = 480$ and $J = 640$ this results to adding 768 pixels at the center. For populated bins this has no noticeable effect on the center of mass, while ensuring bound and well-defined values in case of empty or almost empty bins (cf. figure 6.6d).

6.1.4 (ii) Color Moments

The second set of color-spatial moment features is extracted by splitting the image into subblocks and computing HSV color moments for each of them. No stabilization is necessary for these features because each of the "bins" contains a fixed number of pixels. From the design of the HSV color space, e. g. the hue is undefined at a saturation $S = 0$ and neither hue nor saturation are defined at a value $V = 0$, it is advisory to not compute the moments on the HSV triples directly. Instead, the color space is embedded into the

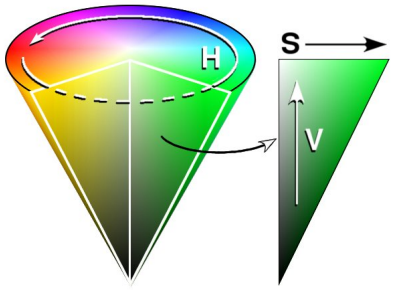

Figure 6.7: HSV color space embedded in $\mathbb{R}^3$ (image source: `de.wikipedia.org/wiki/Bild:HSV_cone.jpg`)

$\mathbb{R}^3$, yielding the typical inverse color cone (figure 6.7).

Definition 6.2. The HSV color cone embedding $\phi(H, S, V)$ in $\mathbb{R}^3$ is defined as

$$\phi(H, S, V) = \begin{pmatrix} SV \cos H \\ SV \sin H \\ V \end{pmatrix}.$$

Having computed the center of mass in the embedding space, its x and y coordinates capture a property that could be called *color dominance*. Knowing that virtual evidence boosting (with the chosen function approximator decision stump) only uses one feature component at a time, having these two values directly as features may be limiting. It would require two learning iterations to capture the dominance of green, while only one would be required for red. To reduce the effects of the standard, but still arbitrary choice of having red at $H = 0°$, the center of mass is described by three "coordinates" with respect to axes at 0°, 120°, and 240° angles to the x axis, instead of the two Cartesian coordinates. The three feature values extracted from the (x, y) coordinates of the center of mass hence are x, $\frac{1}{2}(y - \sqrt{2}x)$, and $\frac{1}{2}(-y - \sqrt{2}x)$.

Furthermore, the first and second moment of the V component are computed by the standard formulae and used as features. And finally the average of the radius in the projection $S \cdot V$ is added as a further feature with the idea to capture the effective color saturation in the image block. With six color moment features for each subblock of and an even 3 by 3 partition this totals in 54 feature values.

sensor stream	sample rate	extracted features
acceleration	3×256 Hz	$4 \times 64 + 7$
audio	16348 Hz	$27 + 38$
barometric pressure	14 Hz	8
visible light (high sampling rate)	128 Hz	25
visible light	3 Hz	14
infrared light	3 Hz	14
relative humidity	1 Hz	8
temperature (from barometer/temperature sensor)	14 Hz	8
temperature (from humidity/temperature sensor)	1 Hz	8

Table 6.1: Multi-sensor board data streams and number of extracted features

6.1.5 Multi-Sensor Board Features

The multi-sensor board combines sensors for audio, visible light, infrared light, temperature, barometric pressure, and humidity as well as a three-way accelerometer. It produces several data streams at very different data rates, from 1/second to 16384/second. To ease the integration into a temporal statistical model, features are extracted at a uniform data rate of 4 Hz, spanning sensor- and feature-specific windows of the input data streams. However all features only use data from the past, so that they would be suitable for an online inference system (cf. section 1.1.1). The feature set is taken from Subramanya et al. [2006] with only a single modification (see the audio features below), so only a brief overview shall be given here. Also see the publication by Lester et al. [2005], who first introduced most of the features.

Several features are based on the immediate values of the data streams, including the mean, (co)variance, differential values, or the response to low-pass filters. For the sensors that have high data rates — the microphone, accelerometers, and the high frequency sampling visible light sensor — the temporal patterns are more interesting. They are captured with several FFT-based features: mean response for frequency coefficient bands with linear or logarithmic band widths, Mel-frequency cepstral coefficients (a feature often used in speech recognition), and spectral entropy and energy. These allow for example to recognize the regular acceleration patters from walking or the 50/60 Hz frequency from fluorescent lamps. As for the accelerometer, the frequency features are computed for the norm of the 3-dimensional acceleration vector and additionally for each component. A further feature is created by integrating the acceleration over windows up to 60 seconds, aiming to capture the gravity vector. The features are typically extracted from longer windows if the data rate is low. This ranges from approximately 0.03 seconds (repeated eight times so that it adds up to 0.25 seconds, the feature cycle time) for some

dataset	# of traces	'inside'	'outside'	'on a bus'
`day`	18	60%	40%	–
`day/bus`	18	56%	37%	6%
`anytime`	24	51%	49%	–
`anyt/bus`	24	43%	41%	15%

Table 6.2: Name and statistics of datasets.

of the audio features up to 10 seconds for humidity and temperature. Unlike in previous work, we don't use the 8×27 audio features from the 0.03 second data windows directly, but average those vectors over the eight repetitions. A summary of the features can be found in table 6.1.

6.2 Recorded Data

For evaluating sensors and features 24 data traces have been recorded with an average length of 27 minutes, totaling to just under 11 hours. The recordings were made during the daily movements of the author and span trips by bus and foot on the University of Washington campus, between campus and home, in the nearby commercial area on 'the Ave', and other parts of the city. The indoor places include the Paul Allan Center and the Husky Union Building on campus, restaurants and shops on the Ave, Northgate Mall, and the home of the author. The recordings were annotated with the true labels while the data was recorded by selecting the current state on a handheld computer. The following state was captured:

Spatial context with the labels 'indoor', 'outdoor', and 'on a bus'.

Mode of movement with the states 'stationary', 'walking', 'going upstairs', 'going downstairs', 'elevator up', and 'elevator down'. These low-level activity labels were captured to get a more challenging task for the learning algorithms (see section 6.3.3). The motion state only refers to the movement performed by the individual, so even on a bus the motion labels are 'walking' and 'stationary'.

The data traces were recorded during different times of the day, most of which in normal daylight (18), but also a few at dawn (2) and at night (4). Since detecting spatial context is obviously harder if there are different lighting conditions in the 'outside' category, we also run tests which only use the traces recorded during the day. Also, in a further simplification, we run tests on data where all instances of the 'in a bus' spatial context

class has been removed by cutting them from the traces. The datasets are summarized in table 6.2.

6.3 Experiments and Results

To evaluate sensors, features, and methods, we train and test conditional random fields on the sequences of feature values computed from the recorded data. The first subsection focuses on comparing camera and multi-sensor board features, and therefore all experiments in that section run the same learning method, namely Liao's VEB. Then, for the method comparison in subsection two, we also report and discuss the (average) accuracies which can be achieved with the other versions of VEB or LogitBoost on the feature comparison experiments. Finally in the third subsection, we present further experiments and results with all methods on more complicated CRF topologies.

Every experiment consist of several runs for leave-one-out cross-validation. For every trace we train a CRF with all other traces, and then test how accurately the labels can be predicted with that CRF on the one "unseen" trace. The average over these repetitions is the reported accuracy. The label prediction is based on marginal probabilities (cf. equation (5.6) on page 40), and the performance measure is the correct classification rate (cf. definition in remark 3.2). The number of training iterations is limited to 15 for both VEB and LogitBoost, and all methods use step functions as function approximators.

For the experiments which only infer spatial context (sections 6.3.1 and 6.3.2), the statistical model is a linear chain conditional random field (cf. figure 5.1 on page 36). We infer the spatial context at the rate of the images, i. e. there is one label variable Y_t for approximately every 1.3 seconds. Consequently we get 5–6 feature vectors from the MSB for every time slice, i. e. pair $(Y_t, \mathbf{X}_t)$. Instead of concatenating these vector and then modeling an accordingly sized input variable $\mathbf{X}_t$, we chose to average the vectors. This may remove some information — transitions may be detected more accurately if the the most recent values were available separately — but also avoids excessively large numbers of features. The model for the joint recognition of activity and spatial context will be presented later in section 6.3.3.

6.3.1 Feature Comparison

For the camera features introduced in section 6.1, there are different versions possible which add or remove some of the feature values. We will first present the results for all feature versions on the smallest subset of our recordings, the `day` dataset, to get an idea which of them merit a further investigation. With the results on all datasets we will first

feature set	accuracy	description	#
hist	81.2%	HSV color histogram (see section 6.1.3)	33
hist+agg	81.7%	ditto with additional aggregate bins	76
hist+centers	89.6%	histogram and bin centers of mass (section 6.1.4 (i))	99
hist+vmean	82.6%	ditto, but only with rows of centers	66
hist+agg+centers	88.0%	histogram with aggregates and bin centers	228
hist+agg+vmean	83.5%	analogously	152
moments mean	82.4%	color moments (1^{st} order moments only)	45
moments	80.6%	color moments as described in section 6.1.4 (ii)	54
moments all	78.6%	ditto with 2^{nd} order color dominance and saturation	90
spyr(4)	75.7%	steerable pyramid (section 6.1.2), 4 scales	384
spyr(4)+row sums	78.6%	ditto with row sums	504
spyr(4)+agg	78.1%	ditto with row and column sums	600
spyr(6)	80.6%	steerable pyramid with all 6 scales	576
spyr(6)+row sums	81.0%	ditto with row sums	756
spyr(6)+agg	81.3%	ditto with row and column sums	900
spyr pca(4, 80)	78.3%	PCA of spyr(4)	80
spyr pca(6, 80)	78.9%	PCA of spyr(6)	80
all cam w/ raw spyr	86.0%	hist+agg+centers, moments, spyr(6)	858
all cam w/ spyr agg	86.5%	ditto, but only with sypr row and column sums	606
all cam w/ spyr pca	86.9%	ditto, but only with PCA of spyr(6)	362

Table 6.3: Test performance for different camera features on the dataset `day` using Liao's VEB. (The last column states the length of the feature vector.)

compare camera features amongst each other, before discussing the question of the best sensor system for recognizing spatial context.

Since virtual evidence boosting performs feature selection, learning should be the more successful the more features the algorithm has to choose from. However, this may be counteracted if overfitting occurs, i. e. the model includes dependencies between features and labels which happen to be present in the training data but are not there in general (cf. remark 3.4). Adding a particular set of features therefore may increase the performance if these features have strong and principled correlations with the labels, but also decrease the performance if they make the learner more susceptible to overfitting. On this background we compare different versions of the camera features presented earlier. Details about their composition and the resulting accuracies on the `day` dataset are shown in table 6.3.

These experiments in particular allows us to draw conclusions about those features which don't belong to all versions of a feature set. For the steerable pyramid features (non-PCA) we see that in particular adding two scales for small structures improves the classification performance. Interestingly, this is not the case for the 'spyr pca' feature sets. Also, the bin centers prove to be a successful addition to the color histograms, especially

feature set	day	day/bus	anytime	anyt/bus
hist+agg	81.7%	75.5%	64.4%	60.8%
hist+centers	**89.6%**	82.0%	78.9%	77.5%
hist+agg+centers	88.0%	82.0%	82.0%	77.5%
moments mean	82.4%	74.7%	71.9%	66.1%
moments	80.6%	77.4%	72.4%	68.3%
spyr(6)	80.6%	76.7%	79.4%	68.1%
spyr(6)+agg	81.3%	78.3%	78.9%	70.2%
spyr pca(6, 80)	78.9%	74.1%	74.8%	67.1%
all cam w/ raw spyr	86.9%	81.9%	84.2%	**84.0%**
all cam w/ spyr agg	86.0%	**82.7%**	81.3%	79.5%
all cam w/ spyr pca	86.5%	81.3%	**86.7%**	80.4%
light	97.4%	92.0%	86.2%	83.1%
audio	95.0%	92.5%	92.6%	88.7%
accel	85.0%	82.0%	82.5%	79.5%
temperature	87.9%	83.4%	85.5%	74.4%
light, audio	**97.8%**	97.0%	91.0%	93.1%
all msb	**97.8%**	**97.5%**	**93.7%**	**95.5%**

Table 6.4: Accuracy in recognizing spatial context with camera and MSB features on all datasets using Liao's VEB.

and contrarily to expectations the horizontal alignment of the centers of mass. Whether the accuracy benefits from the aggregate histogram bins is however rather questionable. The second order color moments actually decrease the performance compared to only using means in this data set. This is different for the following experiments where the variance of the HSV value color channel (the only second order moment in the 'moments' feature set) has an overall positive effect.

For the comparison of the features amongst each other, we also run tests on the more difficult tasks which include the recordings at night and/or the third spatial context label 'on a bus'. The overall results are shown in table 6.4. Furthermore table 6.5 shows from the same experiments the average accuracy on the dusk and night traces only. As for the design of image features for spatial context, we can draw the following conclusions:

- From the color-spatial moment feature sets the histogram with bin center features ('hist+centers') are clearly better than the color moments on image subblocks. Their principal difference is the granularity at which the properties of the pixels — their image position and color — are mapped to feature values (cf. section 6.1.4). The histogram with bin centers uses a fine resolution for the color information (33 bins) and only the first moments for the position (2 values). This contrasts the color

feature set	anytime	anyt/bus
hist+centers	71.3%	63.8%
hist+agg+centers	72.9%	63.3%
moments	58.0%	58.5%
spyr(6)	82.6%	66.2%
spyr(6)+agg	81.3%	65.4%
all cam w/ raw spyr	**83.7%**	76.2%
all msb	**83.7%**	**91.2%**

Table 6.5: Average test result for the traces recorded at dawn or night with model trained on the full dataset (except the test trace). The omitted feature sets yielded lower accuracies.

moment features with 9 bins for position and only 6 moments for color. Since the former performs better, we can conclude the color distribution is more important than the color's positions in the image.[1] Nevertheless, it has to be noted that the position of colors in the image is relevant and that using color-spatial moments instead of only histograms significantly improves the accuracy in recognizing spatial context with a wearable camera.

- As for the direct comparison of steerable pyramid and color features, it is clear that color features (namely both the feature sets 'hist+centers' and 'hist+agg+centers') surpass the performance of only using the former on the daytime traces. This is also true for the average accuracy with the traces recorded at dawn and night included. In particular for these difficult traces however, the data in table 6.5 shows that the steerable pyramid features better cope with the variable lighting conditions in the 'outside' category. Comparing the 'spyr' accuracies for the `anytime` dataset in tables 6.4 and 6.5 even implies that these features slightly benefit from dawn and nighttime situations. While more data would be required to verify this, we can generally agree with the conclusion of Torralba et al. [2003] who state that color features are less suited for recognizing spatial context if the colors vary dramatically within some of the categories. This is interesting insofar as our color features are much more elaborate than the simple features (PCA on raw images) used by Torralba et al.

- For the `day` dataset the best performance is achieved by exclusively using the feature

[1] The color moment features have a disadvantage in this comparison because they are lacking color information which is independent of the image position. A color moment vector describing all pixels in the image should have been included in the feature set for a better comparability. Despite this the given conclusion should be valid.

set 'hist+centers' although these features are also included in all of the combined 'all cam' sets. This shows that VEB may indeed overfit. Using other features than the histogram and bin center features yielded a 1.3 to 2.4 percentage points better training performance (data not shown) but decreased the test performance by to 2.7 to 3.6 points. This allows to draw the general conclusion that despite feature selection there are cases where a manual reduction of the feature set to the most relevant features increases the generalization performance. For the harder datasets this problem is less prominent and the best accuracy (only using the camera sensor) is in fact achieved with a combination of color and steerable pyramid features.

- While the principal component analysis (PCA) significantly reduces the number of steerable pyramid features, it also appears to be removing significant information (if used by themselves). PCA is chosen in the design of features to reduce the dimension of the input to the learning algorithm. However PCA may also counteract the feature selection principle (cf. section 5.4.4). Commonly each principle component combines all entries of the feature vector and so prevents that the learning algorithm only picks those features which clearly have a strong correlation with the spatial context label. A further negative effect of the PCA could be that it reduces the discriminative power of the features since it is computed only based on the variance in the features, ignoring the labels. This cannot be overcome by increasing the number of principal components because VEB rarely even chooses the 50^{th} component or later ones.

  The effect of applying PCA if the steerable pyramid features are used in combination with other camera features however remains inconclusive.

Even with the best camera features, the results in table 6.4 show as expected that the multi-sensor board is generally more suitable for recognizing spatial context than a wearable camera. Especially the light sensors (cf. table 6.1) and the microphone are for obvious reasons very useful for distinguishing the situations inside, outside, and on a bus. With all MSB sensors an enormous correct classification rate of 97.5% is archieved for the daytime traces and a still very impressive 95.5% for all traces.

The feature set 'all msb' has also been used previously by Subramanya et al. [2006] with a minor difference in the audio features (see section 6.1.5). Their best reported accuracy in detecting spatial context without additional input from GPS is 89.4%. It can be assumed that their learning problem was harder because even with identical learning methods, namely LogitBoost, all our datasets can be learned with an accuracy of at least 93.4% (data not shown) and theirs only with 83.8%. In a discussion with the

feature set	all traces	dusk&night only
all cam w/ raw spyr	84.2%	83.7%
all msb	93.7%	83.7%
all msb, hist+agg+centers	94.3%	88.0%
all msb, spyr(6)	**95.0%**	**91.8%**
all msb, all cam w/ raw spyr	94.9%	91.7%

Table 6.6: Accuracy averaged over all traces and only traces recorded at dusk and at night of the `anytime` dataset using Liao's VEB.

authors it was concluded that this may have been due to their more general 'vehicle' category which was harder to detect than the bus context in our data.

An investigation of the accuracies on those traces of the `anytime` dataset which were recorded at dawn or at night (see table 6.5) shows that the MSB features similar to the color features don't generalize particularly well across lighting conditions. This is most prominent in the `anytime` dataset because the bus category is just as easy to detect at night if the MSB's audio sensor is available. While there is barely any room for improvements on the day datasets, combining MSB and camera features can in fact improve the performance on the datasets which include the difficult nighttime traces.[2] Table 6.6 shows that the accuracy is increased by up to 1.3 percent points which can be explained by the better performance on the six dusk and night traces.

To summarize, our experiments confirm that sensors which directly measure properties of the environment are more suitable for recognizing the spatial context categories 'inside', 'outside', and 'on a bus' than a wearable camera. This is not surprising because some of the MSB measurements like high light intensities or diesel engine noises almost directly imply some of the category labels. If however the dependencies get more complex, like in case of the variable light in the outside category, the high-dimensional measurements of the camera sensor turn out to be an advantage. Therefore it can be expected that in a more complex task, e. g. with more spatial context categories, a rich feature set of color-based and color-invariant camera features will be a valuable supplement to the multi-sensor board.

6.3.2 Comparing the Versions of Virtual Evidence Boosting

In this section we present results for the empirical comparison of Liao's virtual evidence boosting and the versions developed in the context of our re-derivation of VEB (section

[2]The accuracies with combined MSB and camera features like in table 6.6 were +0.2% for `day`, −0.2% to −0.4% for `day/bus`, and −0.5% to −0.8% for `anyt/bus` in comparison to "all msb". Overfitting seems to particularly affect camera features in the bus context.

training method	average accuracy	difference
Liao's VEB	83.06%	
VEB with piecewise Newton steps	81.93%	−1.14%
VEB with neighborhood-based Newton steps	81.93%	−1.15%
LogitBoost	80.11%	−2.96%

Table 6.7: Average accuracy of all experiments (except `day/bus` for which the LogitBoost experiments are missing) in the previous section with different learning methods.

5.4). Each experiment for comparing features in the previous section was repeated with all three versions of VEB, but so far only the results for Liao's VEB have been shown. For additional reference we also repeated most experiments (81 out of 100) with LogitBoost as learning method. Note that VEB is equivalent to LogitBoost if the conditional random field doesn't contain any pairwise factors. Also, the VEB versions only differ in how the pairwise factors are learned (see section 5.4.6), so the comparison with LogitBoost points out the specific performance gain from learning label dependencies with each method. We are not comparing VEB to any further learning methods (cf. section 2.2) but refer the reader to the experiments by Liao et al. [2007].

The average accuracy on the same experiments is clearly better with Liao's VEB than with both versions of VEB with Newton steps (see table 6.7). In the direct comparison by experiment, each of our versions only achieve a better performance than Liao's VEB in 13 respectively 14 out of 100 cases. There appears to be no principled pattern behind these cases (see figure 6.8a), so we can conclude that for the linear chain CRFs used in all preceding experiments Liao's VEB is generally the best version of VEB. Our versions still achieve results which are distinctively better than those of LogitBoost (see also figure 6.8b), but with our formulae for learning the pairwise factors, only half of the improvement of Liao's VEB is gained.

As for the comparison of our versions 'VEB with piecewise Newton steps' and 'VEB with neighborhood-based Newton steps' (see section 5.4.3 (iv)), there is no noticeable difference in the resulting accuracies. If the experiments with `day/bus` dataset are included in the averages, the neighborhood-based version is better by 0.04%, but this still rather supports the conclusion. In individual experiments each of them may be better in some cases (see figure 6.8c), but this does not justify the higher computational complexity of VEB with neighborhood-based Newton steps.

In the introduction of the optimization criterion of VEB (section 5.4.2), theoretic analysis yielded that the optimization process for training a CRF may diverge due to approximations. Since all versions of VEB agree on the optimization criterion, all of

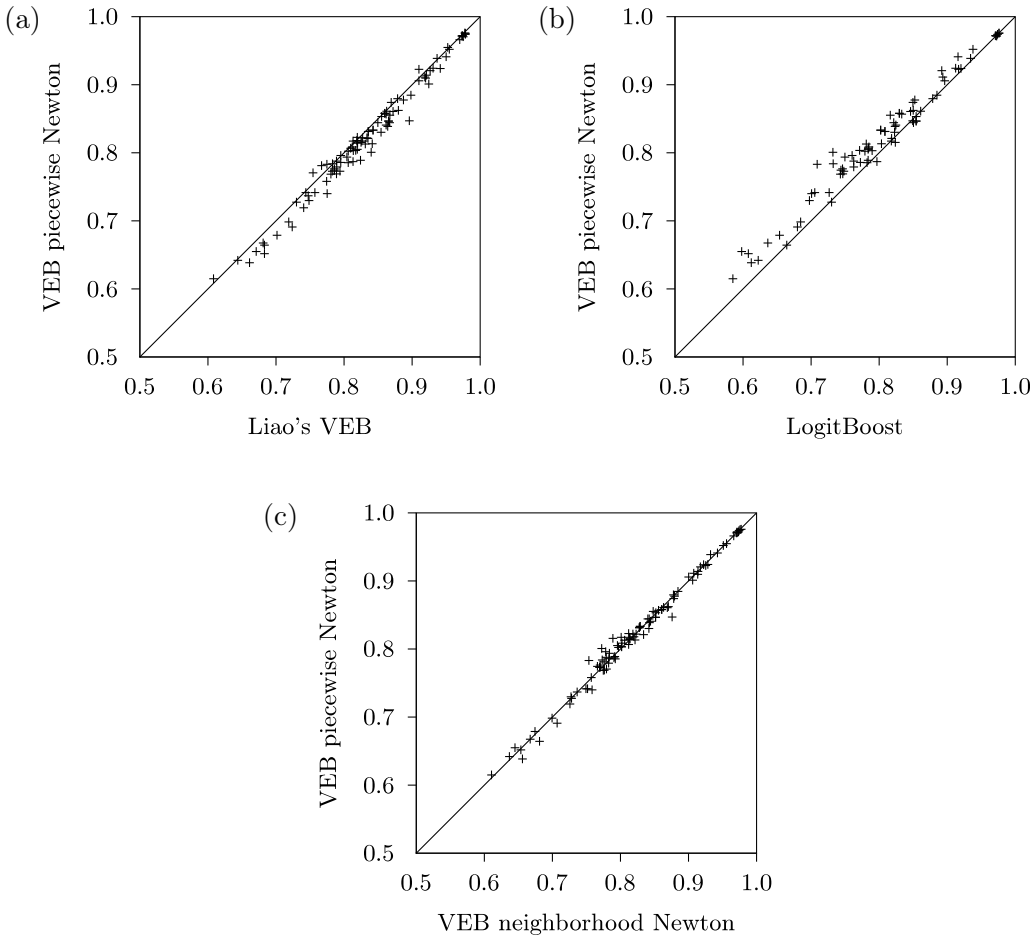

Figure 6.8: (a–b) Accuracy comparison of VEB with piecewise Newton steps to Liao's VEB and LogitBoost. The charts for VEB with neighborhood-based Newton steps (not shown) are very similar. (c) Accuracy comparison of our versions of VEB.

them could be affected by this. In practice this was however not observed on any of the experiments for comparing features. There were only two situations in which the model parameters from the last learning iteration were *not* chosen as the learning result (cf. section 5.4.5). In some cases the last iteration(s) could not further improve the accuracy on the training set, and so the updates of these were discarded by definition. A slightly more alarming situation occurred when the first iteration which updated the (only) pairwise factor template decreased the training performance and the following iterations could not raise the accuracy rate over the high point again. In these cases, the optimization doesn't diverge but rather gets stuck at a suboptimal point in the parameter space. This behavior however only occurred with very uninformative feature sets, like only acceleration or humidity features, and so even if pairwise factors are not learned (like by LogitBoost) no better result can be achieved. Since VEB significantly surpasses LogitBoost in the feature comparison experiments (the latter was more successful than Liao's VEB in only 2 out of 81 experiments), it is clear that the theoretically possible divergence is not an issue on linear chain CRFs.

Having read this conclusion, the reader may ask why the possibility that learning diverges was still emphasized in the derivation of VEB? The analysis of the optimization criterion was motivated by observations made on a preliminary recording (camera system only, not included above) which only consists of a single 25 minute trace split into five parts for cross-validation. Using Liao's VEB and the feature set 'hist+agg', the learning process eventually diverges with increasingly overcompensating updates. The reported accuracy during training increases to approximately 97% around the 13^{th} iteration before decreasing an irregular downward spiral, reaching accuracies as low as 80%. Note that those model parameters are used for testing which achieved the best training accuracy, so just the fact that the algorithm did diverge eventually is not necessarily a problem. The important question is whether the model instance learned before that is accurate and generalizes well. The versions of VEB with Newton steps appear to be more resistant to the divergence issue on that data trace — their accuracies deteriorate by at most 2 percent points after the high point — but the test performance of Liao's VEB, 94.2%, is still much better than the other versions' 89.7% and 89.5% or LogitBoost's 52.3% (!). This implies that even if divergence occurs, it still may not have a significant negative effect on the generally good performance of VEB, in particular Liao's VEB.

6.3.3 Joint Activity Recognition

The experiments in the preceding sections all only inferred a single hidden state, the spatial context, and therefore only required a linear chain CRF. In this section the

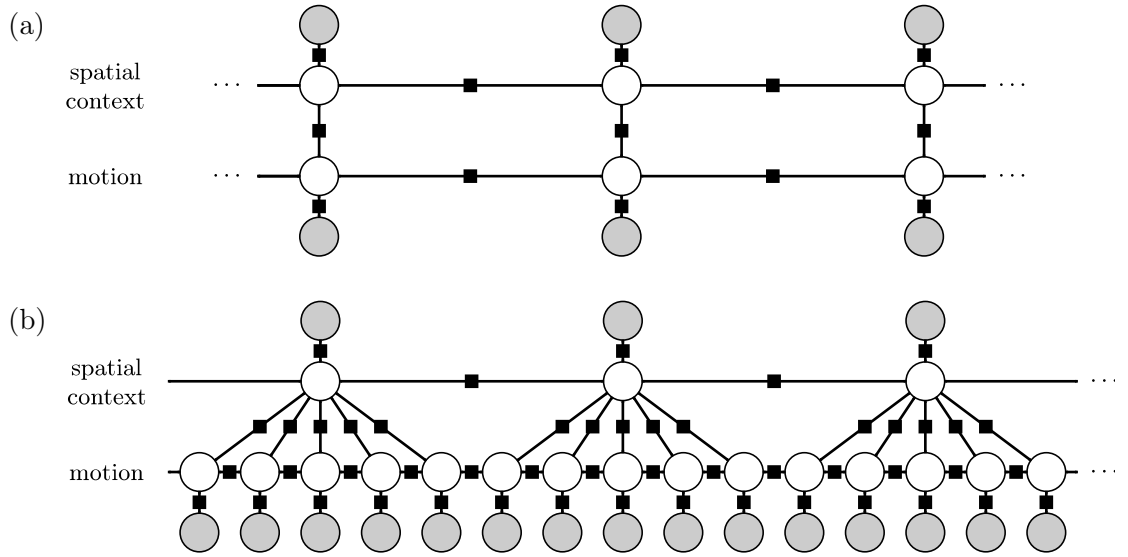

Figure 6.9: CRFs for jointly estimating mode of movement and spatial context with ladder topology (a) and asymmetric topology (b).

versions of VEB will be tested on more challenging topologies for joint estimation of spatial context and mode of movement (cf. section 6.2). Since the motion state changes more often than the spatial context, we investigated two different topologies: one which has a label node each for every captured image (figure 6.9a), and one which estimates the mode of movement at 4 Hz, the rate of the MSB, instead (figure 6.9b). Both CRFs contain two local factor templates and three pairwise factor templates — one each for the temporal relationships of the two state components, and one for connecting the state nodes with each other to represent possible dependencies between activity and spatial context. The observation for each state component is the feature set 'all msb'.

The results for both topologies and all investigated learning methods are shown in table 6.8. For the ladder-shaped CRF these confirm the advantage of Liao's VEB over the other versions. For the asymmetric topology however, our version VEB with piecewise Newton steps achieves a better result than Liao's VEB, yet both methods are surpassed by LogitBoost. An investigation of the algorithms' behavior during training reveals that this is not a coincidence: In that topology, in particular Liao's VEB appears to compute updates to pairwise factors which are too large. After such an updates the accuracy on the training data decreases and the maximum is not reached again in subsequent iterations. Consequently the best iterations in each run are quite early (the median is iteration 7 out of 15) and the best model instances often assumes the state nodes to be independent, i. e. the model matrices of pairwise factors are zero. The training accuracy

method	state	ladder topol.	asym. topol.
Liao's VEB	motion	96.1%	95.2%
	context	96.6%	87.8%
	total	**96.3%**	94.1%
VEB with piecewise Newton steps	motion	95.3%	94.7%
	context	95.2%	92.6%
	total	95.2%	94.4%
VEB with neighborhood-based Newton steps	motion	95.5%	infeasable
	context	95.3%	
	total	95.4%	
LogitBoost	motion	95.4%	95.4%
	context	93.8%	93.8%
	total	94.6%	**95.2%**

Table 6.8: Accuracies for jointly estimating mode of movement and spatial context with different learning methods and topologies for the complete `anyt/bus` dataset. (The model learned by LogitBoost doesn't include any pairwise dependencies.) The input for both states is the feature set 'all msb'.

of VEB with piecewise Newton steps is also not entirely monotonic, but when setbacks occur the accuracy usually recovers (median best iteration is 14 out of 15). Still both these versions of VEB also appear to be "wasting" iterations on learning the pairwise relationships and hence are surpassed by LogitBoost. However it also has to be noted that the good performance of LogitBoost is facilitated by very expressive features on the one hand (the feature set 'all msb'), and on the other hand dependencies between mode of movement and spatial context which can be expected to be rather weak [cf. experiments in Subramanya et al., 2006].

The described issue is unique to the asymmetric CRF shown in figure 6.9b. The key difference of this topology to the other investigated topologies is that each spatial context label node is connected to 5–6 factors with tied parameters, and so learning the parameters for that template appears to be particularly challenging. A principled comparison of VEB to other training methods on complex CRF topologies however requires more than one experiment and remains a task for future research.

It would certainly be interesting to see if VEB with neighborhood-based Newton steps performs better than the other versions of VEB on the asymmetric CRF, since the piecewise approximation, which is used by both other versions, may have a particularly large impact in that topology. However due to the complexity which is exponential in the number of neighboring factors with tied parameters (see section 5.4.3 (iv)), each training run (out of 24 for cross-validation) took more than a day and so the experiment was not

completed. Even if the results were better, it is questionable if they justify the much longer runtime of this version.

The results for the joint recognition of activity and spatial context are again better than previously published accuracies with the same hardware and features and similar or the same learning method [Subramanya et al., 2006, Liao et al., 2007]. This is again due to the fact that we used different data in which not only the spatial context but also the mode of movement appears to be easier to detect. In our recordings the mode of movement is dominated by the categories 'stationary' and 'walking' (56% and 41% of all labels) which are easy to distinguish with the acceleration sensor.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

In this work we provided a detailed introduction to *virtual evidence boosting*, a method for training conditional random fields which has been recently proposed by Liao, Choudhury, Fox, and Kautz [2007]. The introduction also comprises the foundations of VEB: The optimization strategy to learn the parameters of the CRF is based on *LogitBoost*, so we introduced and formally derived that method with a particular focus on steps which remain unclear in the original publication by Friedman et al. [2000]. Key formulae in VEB are taken from *belief propagation*, so we also gave a well-founded introduction to this inference method.

Starting from and closely following the very promising principles of virtual evidence boosting, we provided a completely new derivation of the method, including a close investigation of the underlying optimization criterion. We discovered that the exact updates for pairwise factors computed by Newton's method make the learning fail, but successfully stabilized the optimization process with a heuristic limiting the step size. The original derivation of VEB published in [Liao, 2006] contains an error in the computation of the Newton steps which interestingly yields naturally stable update iterations. Following our correct derivation, we developed two new versions of virtual evidence boosting: *VEB with neighborhood-based Newton steps* and a computational simplification thereof called *VEB with piecewise Newton steps*.

Empirical comparison shows that Liao's VEB exhibits a performance superior to our versions of VEB on linear-chain conditional random fields. Still, all versions of VEB yield a clear performance improvement over LogitBoost by learning label dependencies. A possible issue with using VEB for complex CRF topologies has been discovered but

not fully investigated.

Learning methods like VEB are an important component in human activity recognition applications. We specifically investigated the task of recognizing spatial context with a wearable sensor system, distinguishing the categories 'inside', 'outside', and 'on a bus'. Spatial context is a notion of location which has only rarely been considered in previous work. For our task, we compared the use of a wearable camera with color-based and color-independent features to a multi-sensor board which integrates acceleration, audio, light, temperature, pressure, and humidity sensors. As expected the overall best results are achieved with the MSB sensors. Still, the steerable pyramid camera features are most robust to the enormous lighting difference between day and night within the 'outside' category, and so combining camera and MSB features yields the best results in some of our recordings.

7.2 Future work

While our versions of VEB don't achieve a better performance than the previously published version of VEB, our thorough derivation provides a good basis for future developments. At the moment, we can identify several measures which may yield improved versions of VEB.

- The only stabilization heuristic which has been investigated is the one described in section 5.4.3 (iii). There may be other heuristics which are more suitable.
- Related to this question is the criterion for choosing the factor with the best update (cf. section 5.4.4) because the current formula is based on the Taylor approximation of the update criterion (5.28) and so is affected by heuristics like ours which directly modify the derivatives. It is possible to evaluate the update criterion directly without having to run inference again, so this may yield a better feature selection approach.
- *GentleBoost* by Friedman et al. [2000] is claimed to be more stable than LogitBoost so it may be worth investigating if the GentleBoost optimization criterion can also be applied to training CRFs.

Furthermore there are a few open questions about the performance of VEB in practice. Liao et al. [2007] claim that it is sufficient to run only one iteration of loopy belief propagation in each iteration of VEB. Due to the issues described in section 5.4.6 we always ran belief propagation until convergence, and so an empirical comparison is still

missing. Also, in section 6.3.3 we found that VEB has problems with learning the parameters of a CRFs with a certain complex topologies. We identified the large number of factors with tied parameters connected to a single hidden node as possible reason for this issue, but further experiments would be required to verify this.

Although we investigated several image features which have been successfully used in previous approaches, our comparison did not include the recently very popular local descriptors on salient points in the image (cf. section 2.5). Since these descriptors are only based on grayscales and independent of the position in the image, they may in fact be very suitable for detecting spatial context from a wearable camera and should be investigated in future work. Also it is an interesting open question if a bag of words representation can be directly used as feature vector for VEB or if clustering with *Latent Dirichlet Allocation* [Blei et al., 2003] or similar methods is necessary.

A further interesting type of feature with regard to the changing daylight could be values which depend on the current time. Special care has to be taken to prevent overfitting due to limited training data which does not repeatedly cover all times of the day like in our case. Measures to achieve this could be to only provide discrete time information or by only allowing smooth base functions if continuous time features are used. An easy solution could also be to take the difference of the measured light intensity and a reference value for the currently expected daylight and to use the result as feature value. This also remains a task for future work.

Bibliography

Daniel Ashbrook and Thad Starner. Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In *Proceedings of the 2nd International Conference on Pervasive Computing*, pages 1–17. Springer, 2004.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *Proc. of the 9th European Conference on Computer Vision*, volume 1, pages 404–417. Springer, 2006.

Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.

Leo Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7): 1493–1517, 1999.

Yu-Chung Cheng, Yatin Chawathe, Anthony LaMarca, and John Krumm. Accuracy characterization for metropolitan-scale Wi-Fi localization. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications and Services*, pages 233–245, 2005.

DARPA. Proposer information pamphlet for Advanced Soldier Sensor Information System and Technology (ASSIST). Technical Report BAA 04-38, Information Processing Technology Office (IPTO) Defense Advanced Research Projects Agency (DARPA), 2004. Available at `http://webext2.darpa.mil/baa/pdfs/baa04-38PIP.pdf`.

Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the 7th National Conference on Artificial Intelligence (AIII)*, pages 524–529, 1988.

Li Fei-Fei and Pietro Perona. A Bayesian hierarchical model for learning natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 524 – 531, 2005.

Brian Ferris, Dirk Hähnel, and Dieter Fox. Gaussian processes for signal strength-based location estimation. In *Proceedings of Robotics: Science and Systems II*, August 2006.

Marcus Frean and Tom Downs. A simple cost function for boosting. Technical report, Department of Computer Science and Electrical Engineering, University of Queensland, 1998.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

Tarak Gandhi and Mohan M. Trivedi. Pedestrian collision avoidance systems: a survey of computer vision based recent studies. In *Proceedings of 9th International IEEE Conference on Intelligent Transportation Systems*, pages 976–981, 2006.

D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999.

Charles J. Geyer and Elizabeth A. Thompson. Constrained monte carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society*, 54(3):657–699, 1992.

Luc J. Van Gool, P. Dewaele, and André Oosterlinck. Texture analysis anno 1983. *Computer Vision, Graphics, and Image Processing*, 29(3):336–357, 1985.

Monika M. Gorkani and Rosalind W. Picard. Texture orientation for sorting photos "at a glance". In *Proceedings of the IEEE Conference on Pattern Recognition*, 1994.

Trevor Hastie and Robert Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

Jeffrey Hightower. From position to place. In *Proceedings of the 2003 Workshop on Location-Aware Computing*, pages 10–12, 2003.

Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian Smith, and Jeff Hughes. Learning and recognizing the places we go. In *UbiComp 2005: Ubiquitous Computing*, pages 159–176. Springer, 2005.

Joyce Ho and Stephen S. Intille. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of CHI 2005 Connect: Conference on Human Factors in Computing Systems*, pages 909–918. ACM Press, 2005.

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57, 1999.

Stephen S. Intille, John Rondoni, Charles Kukla, Isabel Iacono, and Ling Bao. A context-aware experience sampling tool. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, pages 972–973. ACM Press, 2003.

Timor Kadir and Michael Brady. Saliency, scale and image description. *International Journal of Computer Vision*, 45(2):83–105, 2001.

Steven M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*, chapter 7. Prentice Hall, 1993.

Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

Michael J. Kearns, Ming Li, and Leslie G. Valiant. Learning Boolean formulae. *Journal of the ACM*, 41(6):1298–1328, 1995.

Nicky Kern, Bernt Schiele, Holger Junker, Paul Lukowicz, and Gerhard Tröster. Wearable sensing to annotate meeting recordings. *Personal and Ubiquitous Computing*, 7(5): 263–274, 2003.

Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. American Mathematical Society, 1980.

Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence*. Series in Computer Science and Data Analysis. Chapman & Hall/CRC, 2004.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.

Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place lab: Device positioning using radio beacons in the wild. In *Proceedings of the 3rd International Conference on Pervasive Computing*, pages 116–133. Springer, 2005.

Zhang Lei, Lin Fuzong, and Zhang Bo. A CBIR method based on color-spatial feature. In *Proceedings of the IEEE Region 10 Conference (TENCON 99)*, volume 1, pages 166–169, 1999.

Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 766–772, 2005.

Thomas Leung and Jitendra Malik. Recognizing surfaces using three-dimensional textons. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 1010–1017, 1999.

Lin Liao. *Location-Based Activity Recognition*. PhD thesis, University of Washington, 2006.

Lin Liao, Dieter Fox, and Henry A. Kautz. Learning and inferring transportation routines. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, pages 348–353, 2004.

Lin Liao, Dieter Fox, and Henry Kautz. Location-based activity recognition using relational markov networks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 773–778, 2005.

Lin Liao, Tanzeem Choudhury, Dieter Fox, and Henry Kautz. Training conditional random fields using virtual evidence boosting. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2530–2535, 2007.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 1150–1157, 1999.

Jianchang Mao and Anil K. Jain. Texture classification and segmentation using multiresolution simultaneous autoregressive models. *Pattern Recognition*, 25(2):173–188, 1992.

Natalia Marmasse and Chris Schmandt. A user-centered location model. *Personal and Ubiquitous Computing*, 6(5/6):318–321, 2002.

Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus Frean. Functional gradient techniques for combining hypotheses. In Peter J. Bartlett, Bernhard Schölkopf, Dale Schuurmans, and Alex J. Smola, editors, *Advances in Large-Margin Classifiers*, chapter 12. MIT Press, 2000.

David Mease and Abraham Wyner. Evidence contrary to the statistical view of boosting. Available at `www.davemease.com/contraryevidence/`, 2005.

Ron Meir and Gunnar Rätsch. *An introduction to boosting and leveraging*, pages 119–184. Lecture Notes in Computer Science. Springer, 2003.

Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2):43–72, 2005.

Tom Minka. Discriminative models, not discriminative training. Technical report msr-tr-2005-144, Microsoft Research, 2005.

Yu-Ichi Ohta, Takeo Kanade, and Toshiyuki Sakai. Color information for region segmentation. *Computer Graphics and Image Processing*, 13(3):222–241, 1980.

Aude Oliva and Antonio B. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42 (3):145–175, 2001.

Greg Pass and Ramin Zabih. Comparing images using joint histograms. *Multimedia Systems*, 7(3):234–240, 1999.

Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. In *Proceedings of the 4th ACM International Conference on Multimedia*, pages 65–73, 1996.

Donald J. Patterson, Lin Liao, Krzysztof Gajos, Michael Collier, Nik Livic, Katherine Olson, Shiaokai Wang, Dieter Fox, and Henry A. Kautz. Opportunity Knocks: A system to provide cognitive assistance with transportation services. In *UbiComp 2004: Ubiquitous Computing*, pages 433–450. Springer, 2004.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

Alex Pentland. Smart rooms. *Scientific American*, 274(4):68–76, 1996.

Matthai Philipose, Kenneth P. Fishkin, Mike Perkowitz, Donald J. Patterson, Dirk Hähnel, Dieter Fox, and Henry Kautz. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

Rosalind W. Picard and Monika M. Gorkani. Finding perceptually dominant orientations in natural textures. *Spatial Vision*, 8(2):221–253, 1994.

Ingmar Posner, Derik Schroeter, and Paul Newman. Using scene similarity for place labeling. In *Proceedings of the International Symposium on Experimental Robotics*, 2006.

Pedro Quelhas, Florent Monay, Jean-Marc Odobez, Daniel Gatica-Perez, Tinne Tuytelaars, and Luc J. Van Gool. Modeling scenes with local descriptors and latent aspects. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, pages 883–890, 2005.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Trygve Randen and John Håkon Husøy. Filtering for texture classification: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4): 291–310, 1999.

Todd R. Reed and J. M. Hans du Buf. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Understanding*, 57(3):359–372, 1993.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):277–336, 1999.

Robert E. Schapire, Yoav Freund, Peter Barlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B. Cremers. People tracking with a mobile robot using sample-based joint probabilistic data association filters. *International Journal of Robotics Research*, 22(2), 2003a.

Dirk Schulz, Dieter Fox, and Jeffrey Hightower. People tracking with anonymous and ID-sensors using Rao-Blackwellised particle filters. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 921–926, 2003b.

Eero P. Simoncelli and William T. Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Proceedings of the 2nd IEEE International Conference on Image Processing*, volume 3, pages 444–447, 1995.

Maneesha Singh and Sameer Singh. Spatial texture analysis: a comparative study. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 1, pages 676–679, 2002.

Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 1470–1477, 2003.

Alvy Ray Smith. Color gamut transform pairs. In *SIGGRAPH 78, Conference Proceedings*, pages 12–19, 1987.

Joshua R. Smith, Kenneth P. Fishkin, Bing Jiang, Alexander Mamishev, Matthai Philipose, Adam D. Rea, Sumit Roy, and Kishore Sundara-Rajan. RFID-based techniques for human-activity detection. *Communications of the ACM*, 48(9):39–44, 2005.

Markus Stricker and Alexander Dimai. Color indexing with weak spatial constraints. In *Storage and Retrieval for Image and Video Databases IV*, pages 29–40, 1996.

Markus Stricker and Markus Orengo. Similarity of color images. In *SPIE Conference on Storage and Retrieval for Image and Video Databases*, pages 381–392, 1995.

Amarnag Subramanya, Alvin Raj, Jeff Bilmes, and Dieter Fox. Recognizing activities and spatial context using wearable sensors. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, 2006.

Shamik Sural, Gang Qian, and Sakti Pramanik. Segmentation and histogram generation using the HSV color space for image retrieval. In *Proceedings of the International Conference on Image Processing*, pages 589–592, 2002.

Charles Sutton and Andrew McCallum. Piecewise training of undirected models. In *21st Conference on Uncertainty in Artificial Intelligence*, 2005.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006.

Charles Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the 24th International Conference on Machine Learning*, pages 863–870, 2007.

Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

Martin Szummer and Rosalind W. Picard. Indoor-outdoor image classification. In *International Workshop on Content-Based Access of Image and Video Databases*, pages 42–51, 1998.

Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Textural features corresponding to visual perception. *IEEE Transactions on Systems, Man and Cybernetics*, 8(6): 460–473, 1978.

Antonio Torralba, Kevin P. Murphy, William T. Freeman, and Mark A. Rubin. Context-based vision system for place and object recognition. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, volume 2, pages 273–280, 2003.

Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Contextual models for object detection using boosted random fields. In *Advances in Neural Information Processing Systems 17*, 2004.

Aditya Vailaya, Anil K. Jain, and Hong Jiang Zhang. On image classification: city images vs. landscapes. *Pattern Recognition*, 31(12), 1998.

Aditya Vailaya, Mário A. T. Figueiredo, Anil K. Jain, and Hong-Jiang Zhang. Image classification for content-based indexing. *IEEE Transactions on Image Processing*, 10 (1), 2001.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

Julia Vogel and Bernt Schiele. Natural scene retrieval based on a semantic modeling step. In *Proceedings of the 3rd International Conference on Image and Video Retrieval*, pages 207–215, 2004.

Danny Wyatt, Tanzeem Choudhury, and Henry Kautz. Capturing spontaneous conversation and social dynamics: A privacy-sensitive data collection effort. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, 2007.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. In Gerhard Lakemeyer and Bernhard Nebel, editors, *Exploring Artificial Intelligence in the New Millennium*, chapter 8, pages 239–269. Morgan Kaufmann, 2002.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

Elaine C. Yiu. Image classification using color cues and texture orientation. Master's thesis, Massachusetts Institute of Technology, 1996.