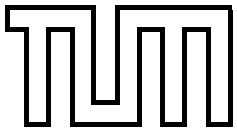
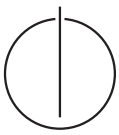


TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK



Intelligent Autonomous Systems Group

Learning Where Objects Are – Organizational Principles in Human Environments

Das Lernen von Aufbewahrungsorten – Organisationsprinzipien in menschlichen Umgebungen

Master's Thesis in Informatik

Author:	Martin J. Schuster, M.Sc.
Supervisor:	Prof. Michael Beetz, Ph.D.
Advisors:	Dipl-Inf. Dominik Jain
	Dipl-Ing. Moritz Tenorth
	Dejan Pangercic, M.Sc.
Submission Date:	27.04.2011

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis only supported by declared resources.

München, den 27.04.2011

Martin Schuster

Zusammenfassung

Für unterstützende Robotersysteme stellt der Transport von Objekten in menschlichen Alltagsumgebungen eine wichtige Aufgabe dar, für die auf der technischen Ebene erste zweckmäßige Lösungen existieren. Als eine von vielen komplexen Inferenzaufgaben gewinnt dabei die Fragestellung, wo Objekte aufzunehmen und wo sie abzustellen sind, zunehmend praktische Bedeutung. In dieser Arbeit betrachten wir die Identifizierung einer Organisationsstruktur in menschlichen Umgebungen, d.h. die Frage nach Organisationsprinzipien, welche es einem Roboter erlauben zu inferieren, an welchem Ort er am besten ein bestimmtes, neues Objekt platzieren oder nach Objekten eines bestimmten Typs suchen sollte, gegeben vergangene Beobachtungen über die Platzierung anderer Objekte in der Umgebung. Diese Fragestellung kann als Klassifizierungsaufgabe formuliert werden. Wir sind der Auffassung, dass Organisationsprinzipien durch einen Begriff von Ähnlichkeit zwischen Objekten bestimmt sind und präsentieren eine empirische Analyse der Wichtigkeit verschiedener solcher Merkmale in Datensätzen, welche die Organisationsstruktur in Küchen beschreiben. Um Ähnlichkeiten als kontinuierliche Features in probabilistischen Frameworks modellieren zu können, erweitern wir zwei probabilistische Modellierungsmethoden, den naive Bayes Klassifikator und Markov Logic Networks, um ihre Parameter mit weichen Evidenzen lernen zu können. Wir vergleichen verschiedene Methoden zur Lösung der oben genannten Klassifiktationsaufgabe und erreichen dabei durchschnittliche Genauigkeiten von mindestens 79% in allen Szenarien. Wir zeigen dadurch, dass sich Ontologie-basierte Ähnlichkeitsmaße gut als hochdiskriminative Features für diese Aufgabe eignen. Wir haben unsere erfolgreichsten Klassifikatoren in das KnowRob Wissensverarbeitungsystem integriert und stellen damit eine Open Source Implementierung bereit, die als Teillösung für komplexe Inferenzaufgaben in Robotersystem einfach zu integrieren ist.

Abstract

In the context of robotic assistants in human everyday environments, pick and place tasks are beginning to be competently solved at the technical level. The question of where to place objects or where to pick them up from, among other higher-level reasoning tasks, is therefore gaining practical relevance. In this work, we consider the problem of identifying the *organizational structure* within an environment, i.e. the problem of determining organizational principles that would allow a robot to infer where to best place a particular, previously unseen object or where to reasonably search for a particular type of object given past observations about the allocation of objects to locations in the environment. This problem can be reasonably formulated as a classification task. We claim that organizational principles are governed by the notion of similarity and provide an empirical analysis of the importance of various features in datasets describing the organizational structure of kitchens. In order to model similarities as continuous features within probabilistic frameworks, we extend two probabilistic modeling methods, naive Bayes classifiers and Markov Logic Networks, to handle soft evidence during parameter learning. For the aforementioned classification tasks, we compare classification methods, reaching average accuracies of at least 79% in all scenarios. We thereby show that, in particular, ontology-based similarity measures are well-suited as highly discriminative features. We integrated the most successful variants of our new algorithm into the KnowRob knowledge processing system to provide an open source solution that is easily applicable for high-level reasoning tasks in robotic systems.

Acknowledgments

I thank Prof. Michael Beetz and my advisors, Dominik Jain, Moritz Tenorth and Dejan Pangercic for their great support throughout this work. We thank Prof. Charlie Kemp for many helpful discussions on organizational principles, our colleagues who provided kitchen images for our analysis and everyone who contributed to our kitchen datasets.

Contents

List of Figures

1 Introduction

As many of the more fundamental problems in robotics (e.g. with regard to perception, navigation and motion planning) are being solved to a degree where we can consider the respective components to be sufficiently reliable to form the basis for complex tasks, the high-level reasoning capabilities of robots will become increasingly important in the years to come.

In view of aging societies in many countries, the field of service robotics in general and the field of robotic household assistants in particular demands our attention. Whereas until now most robotics applications were restricted to well-defined industrial or laboratory environments, recent advances will allow new generations of robots to leave these well-structured places and enter human-centered environments. This development started in the recent years with low-cost robots like the Roomba[1] that perform rather simple tasks such as vacuuming the floor in artificially restricted spaces. But as research advances and hardware prices continue to drop, more complex applications for autonomous assistive robots in human environments are possible.

One reason why most of these robots are not on the market yet is that human environments still pose a huge challenge to state-of-the-art robot systems. Perception and actuation in these unstructured and only partially-known environments are inherently difficult, but for competent behaviour in human everyday environments to become a reality, also a deep understanding of human environments and the interactions between entities within them is a necessary precodondition. Robots will have to reason about everyday environments, the containers and objects these are likely to contain, the properties and functions of objects and their relationships to tasks and other objects in order to achieve competent problem-solving behaviour.

A multitude of research questions arise, out of which only a tiny subset has yet been addressed. With pick and place tasks being addressed by many researchers at the technical level, the question of where to place objects or where to pick them up from, among other higher-level reasoning tasks, gains practical relevance in robotic applications.

[1] see http://www.irobot.com/

1.1 Organizational Structure in Kitchen Environments

In this work, we consider the problem of identifying the *organizational structure* within an environment, i.e. the problem of determining organizational principles that would allow a robot to infer where to best place a particular, previously unseen object or where to reasonably search for a particular type of object given knowledge about the object type and past observations about the allocation of objects to locations in the environment. In particular, we consider kitchen environments in which various utensils, food preparation devices, foodstuffs and food ingredients are typically assigned to storage locations such as cupboards, drawers, refrigerators and working surfaces, see Figure 1.1.

Given a set of previously observed objects with their associated storage locations, the robot is to acquire models that would allow it to, for instance, reasonably allocate each of the objects it might find within a shopping bag to appropriate storage locations. Another task would be to infer the most probable location(s) where to find a specific object, given a model based on observations of the locations of other objects in the environment. These observations could, for example, have been made at a previous point in time when the object in question was not yet present.

As far as the granularity at which we solve the problem is concerned, we distinguish only between the various storage locations but not the placement within these storage locations. We leave the latter as a topic for future research, as it is sometimes relevant in real-world environments. Our analysis of organization principles showed that, for example, some people prefer to store foodstuffs that will expire sooner at more accessible front locations.

We believe that for an organizational principle to be discernable (to robots or humans) at any given location, the objects placed at the location must share certain

Figure 1.1: Examples for groups of objects found in several kitchens

characteristics, i.e. the entropy of the random variable describing the distribution of object properties – in some attribute space – should ideally be low. In order to identify an organizational principle, we must either make direct use of a suitable attribute space or consider an aggregate in the form of a similarity measure. At the global level, an organizational structure will be identifiable if the attribute subspaces pertaining to any two locations are sufficiently different from each other.

1.2 Ontologies

A robot system that is to learn and reason about the structure of its environment and the objects located therein requires large amounts of background knowledge about all the objects that it could potentially encounter. Such background knowledge can be conveniently represented within an ontology containing concepts (classes) for all the relevant types of objects as well as their attributes and relations between objects.

We use the KnowRob ontology [1], which incorporates parts of the OpenCyc upper ontology[2] and extends it with more detailed knowledge about the household domain. For the task at hand, we require the ontology to contain additional information about a multitude of manufactured products found in kitchen environments, particularly foodstuffs. Manually encoding such knowledge for hundreds of objects is a tedious task, but it can be automated: Online shops provide very similar information, though not represented as an ontology. We use a system that automatically translates the category structure of a shopping website, in our example *germandeli.com*, into a class taxonomy in the knowledge base[2]: For example, *Dallmayr Prodomo Coffee* is represented as a sub-class of *Dallmayr coffee*, *Coffee (German Brands)*, *Beverages*, and finally *Groceries*. Using only the *germandeli.com* website, we obtained an ontology that extends the KnowRob ontology with knowledge about more than 7,000 manufactured products (see Figure 1.2 for a small excerpt).

In addition to the category structure, online shops also provide detailed descriptions of the properties of products, such as the perishability status, price, ingredients, etc. This information is usually presented in a semi-structured way in the form of tables or symbols, which can automatically be translated into attributes of the respective object classes. The product images found on the product page can be used to construct recognition models that allow the robot to both detect these objects and to reason about their properties [3].

[2] The translation engine and the generated ontology are publicly available as open-source software at http://code.in.tum.de/pubsvn/knowrob/tags/latest/comp_germandeli

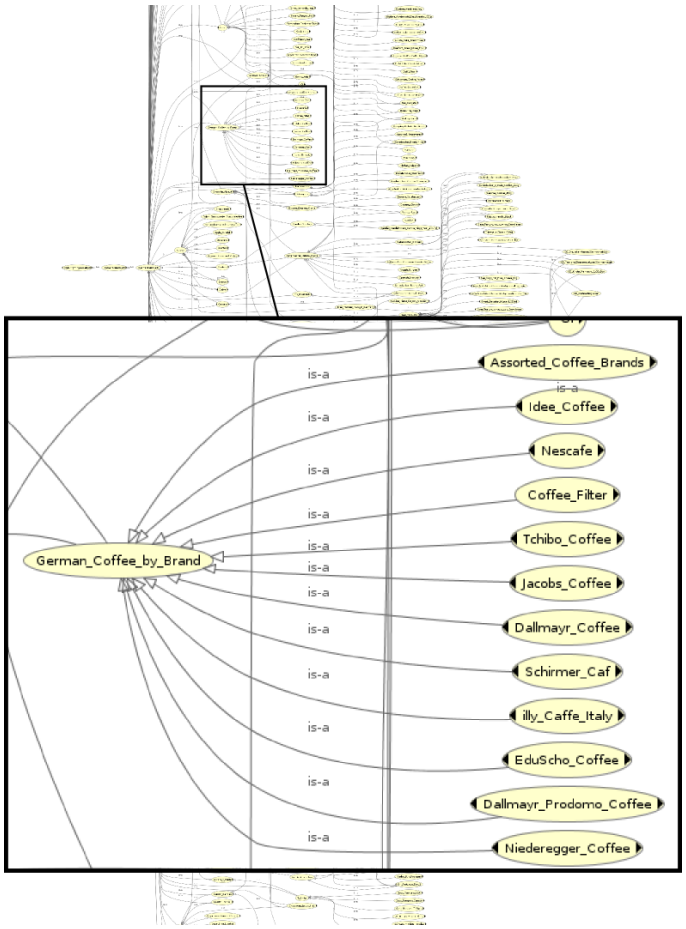


Figure 1.2: Excerpt of the ontology generated from the category structure of the *germandeli.com* shopping website.

1.3 Contributions

Our key contributions in this work fall into two categories. First, we present extensions to two probabilistic machine learning techniques to handle soft evidence:

- an extension of naive Bayes classifiers for soft evidence inference and soft evidence parameter learning;
- an extension of Markov Logic Networks for soft evidence parameter learning, providing several new weight learning algorithms;
- the implementation of the Markov Logic Network learning algorithms within the *Probcog* [4] framework;

Second, we applied these methods (among others) to model the organizational principles within a kitchen environment. As key contributions, we present:

- the selection of suitable attributes and similarity measures derived from an ontological knowledge base, which we partly build up based on web resources;
- an analysis of the performance of various modelling and classification schemes with regard to the aforementioned allocation problem;
- an analysis of the degree to which organizational principles can be identified at the locations within the kitchens we considered, and the degree to which there is a global structure that sets apart the locations;
- an open source implementation of our algorithm to solve the object allocation problem, integrated into the KnowRob [1] knowledge processing system.

1.4 Related Work

There is a large body of related work that considers organizational principles at a fairly coarse level, seeking to exploit knowledge about object-room associations for the purpose of room classification in the context of mapping and navigation [5] or, inversely, visual search for objects given the classes of rooms within a map [6], or both [7]. Object-Room associations are typically described using logical knowledge, e.g. represented in a description logics knowledge base [8, 6], and/or using conditional probability distributions [7, 9].

The problem we consider is qualitatively different, because we do not assume that there is a given, globally applicable principle according to which an environment is likely to be structured, regardless of, for example, the personalities and preferences of the environment's inhabitants. Instead, we are interested in recovering principles based on observations of a single environment in order to generate a model that reflects its idiosyncrasies. Moreover, we disregard the types of containers/rooms and therefore do not establish object-container relationships but rather object-object relationships that induce local clusterings. We thereby want to capture the unique characteristics of each place instead of assigning concepts to places [10] which are usually defined by a single criterion, e.g. by the concept "Bottle-Group".

The authors of [11] use pairwise similarities between household objects to distinguish between object categories in the context of a odd one out task, using features created from auditory and proprioceptive sensory feedback of their robot. This

shows that "natural" object categories (which correspond to higher-level classes in a taxonomy) are related to the perceptual similarities between objects, so we can regard the similarity between classes of objects within a taxonomy as an aggregate of perceptual similarities.

1.5 Overview

In the following section, we present an analysis of organizational principles in the context of kitchen environments. Section 3 is concerned with the datasets we used to evaluate our methods. In Section 4 we describe two probabilistic modeling methods, the naive Bayes model and Markov Logic Networks, and introduce our extensions that allow these models to handle soft evidence. In Section 5 we present our methods to model the kitchen environment, including definitions of the features we use for the classification task of allocating objects to locations and definitions of measures for the importance and discriminative power of the various features. We discuss the evaluation of our approach on the aforementioned datasets in Section 6. In Section 7 we describe the integration of our algorithm within the KnowRob knowledge processing system and its usage in the context of solving a higher-level robot task. In the final section, we draw conclusion about the applicability and discriminative power of our similarity measures and give an outlook on interesting topics for future work.

2 Organizational Principles

To acquire a notion of what might constitute an organizational principle in real-world kitchen environments, we analyzed photographs of kitchens as well as blogs[1] and videos from the Internet. Focusing on the questions of where objects are located, which objects are grouped together, and why, our analysis of this data led to the following prevalent organizational principles:

- *Class:* Most places contain objects that belong to similar classes as they might appear in a taxonomy. For example, there is often a distinction between food and non-food items. More specifically, most people store, for example, prepared food, ingredients, spices, dishes, cutlery or kitchen utensils at separate locations (see e.g. Figure 1.1 and Figure 2.1).

- *Physical Constraints:* Objects are often placed with respect to constraints imposed by their physical properties. For example, large items can obviously be placed only at locations that provide sufficient space, perishable items are stored in a fridge or freezer, objects that can easily be stacked (e.g. different kinds of plates, see Figure 1.1 and Figure 2.1) are placed on top of each other.

- *Purpose:* Objects are often grouped according to the purpose they serve. For example, sugar and coffee beans are used to make coffee and therefore may be placed close together. Similarly, all ingredients used for baking are often found in the same place. In Figure 1.1, objects needed to prepare hot drinks are grouped together (e.g. teabags, coffee and coffee filters).

We also discovered the following additional organizational principles but found them to be less relevant:

- *Packaging:* Oftentimes, large packs of products not intended for daily use are kept in stock, located at different places than single products intended for immediate use. For instance, a single bottle of beer may be kept in the fridge while a crate may be stored elsewhere.

[1] e.g. http://cakescraps.wordpress.com/2010/01/02/organizing-your-fridge/ and http://www.beruly.com/?p=279

Figure 2.1: Further examples for groups of objects found in several kitchens

- *Safety:* Some people place items in lower compartments because they might cause injury as a result of dropping from a high location. Similarly, food items are not usually stored together with items that could spoil them.

The principles pertaining to class, physical constraints and packaging were also considered to be relevant for object placement in grocery stores [12].

All the above criteria can be reasonably translated into similarity measures between pairs of objects.

3 Datasets

We gathered data about the organization within twelve different kitchen environments. Ten of these were acquired by simulating the process of placing objects within a fictitious kitchen, two were obtained by carefully annotating the object locations in two real kitchens. We divided each kitchen environment into *locations*, a location representing a container or tabletop, e.g. a cupboard, a drawer, the fridge, etc.

3.1 Kitchen Mockups

From our extended KnowRob ontology, we selected 66 different concepts to define an exemplary kitchen inventory with a total of 152 objects (instances of concepts). We printed each concept including a product image and the number of its instances on a small piece of paper. We also printed two different sketches of kitchen layouts on large sheets of paper, marking the available containers (cupboards, drawers, fridge) with numbers indicating locations where objects may be placed. The first kitchen mockup had twelve different locations, the second six. We then asked ten persons, five for each of the two layouts, to place the 66 pieces of paper representing the 152 objects at the different locations, grouping them together as if they were to establish an order in their own new kitchen. Two examples from these datasets can be seen in Figure 3.1 and Figure 3.2.

We then annotated the mapping of products to locations for each of our mockup kitchen datasets D_{m1} to D_{m10} in a database that we used for our evaluation (see Section 6). The layout itself served only as a visual aid for our test subjects, potentially improving their impression of acting in a real kitchen and therefore the quality of the data. We did not use information about the proximity of different places or the proximity of places to devices like the oven or sink, although this information might give further clues for reasonable object placement and could constitute an interesting subject for future research.

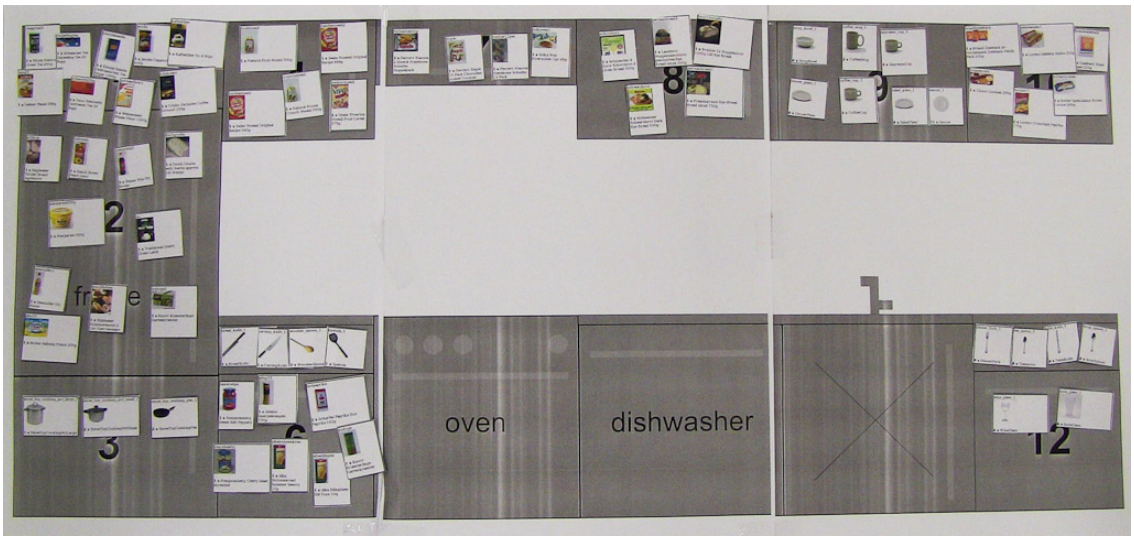


Figure 3.1: Large kitchen mockup with objects placed at twelve different locations

Figure 3.2: Small kitchen mockup with objects placed at six different locations

3.2 Real Kitchens

We gathered one additional dataset from a real kitchen, where we manually annotated all objects along with the location at which they were placed. We then added any missing product classes to our kitchen ontology. We show a visualization of a small excerpt of it in Figure 3.3. Our real kitchen datasets D_{r1} and D_{r2} contain 166 and 87 different classes with totals of 408 and 149 objects placed at 19 and 15 different locations respectively. In the real kitchens, not all objects belonging to a single class were placed at the same location, because the owners distinguished between different states of objects, e.g. partially used products and new products which do not require cooling. In our dataset, we did not consider this distinction, moving all objects of the same class to a single location (less than seven objects relocated in each dataset).

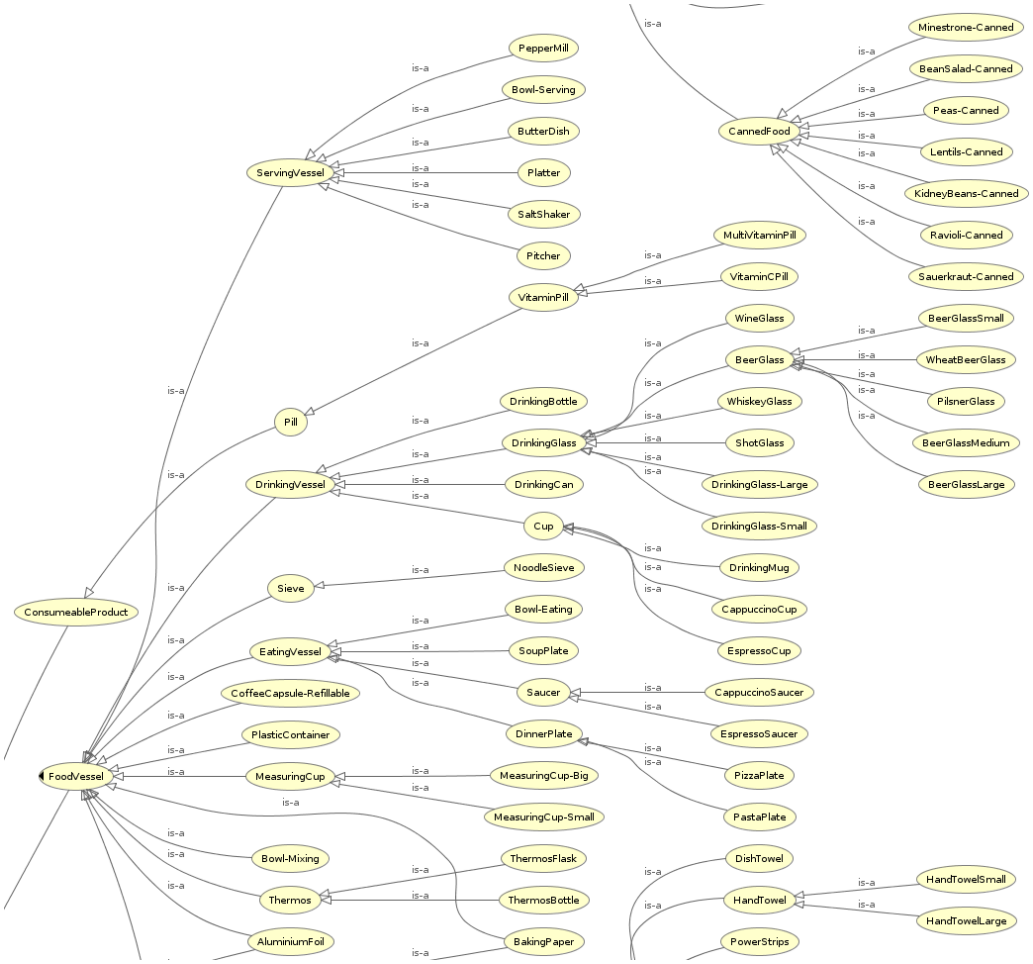


Figure 3.3: Small excerpt from our kitchen ontology, containing concepts used in our real kitchen datasets

4 Probabilistic Modeling Methods

In this section, we present two probabilistic models, the naive Bayesian model (Section 4.3) and Markov Logic Networks (Section 4.4), which we want to apply to model our kitchen environment and solve the classification task therein of allocating objects to locations.

Probabilistic models have several advantages over other classifiers like decision trees or support vector machines. They give us not only a single *winner* class but a probability distribution over all classes. In our kitchen scenario, this can be useful if the classification is only a subtask in a larger probabilistic framework, in which the final placement location can depend on other factors than optimality of the placement itself. We could, for example, consider all locations with a probability exceeding a certain threshold as sufficiently good and then choose the one that is easiest to reach for the robot or the one that provides the most free space within.

We will only consider generative probabilistic models here. They enable us to solve other additional challenges than just the classification task. The naive Bayesian classifier for example can give us insight in the organizational structure present at a particular location by looking at the probabilities of the feature values, given the location. Markov Logic Networks are able to perform joint inference for the locations of multiple objects. Complex models might be applied to answer further questions that could be of interest to an assistive robot acting in a kitchen environment, for example the inference of the types of containers, given their content and spatial relations with respect to each other.

We extend the naive Bayesian model and Markov Logic Networks to handle soft evidence during parameter learning and apply this extension as one way to handle similarities as continuous features within these probabilistic frameworks.

4.1 Probabilities and Degrees of Truth

We want to incorporate continuous features in our probabilistic frameworks in order to model measures of similarity. It is therefore important to be aware of different ways how to interpret continuous variables in that context. Assume we have a proposition R: "The objects O1 and O2 are similar". We now have to distinguish between the *probability* $P(R) \in [0; 1]$ to which this statement is true and the *degree of truth* $T(R) \in [0; 1]$ of this statement. If on the one hand we look only at the *probability*, we treat the statement as boolean and $P(R) = 1$ would mean that the statement is certainly true, $P(R) = 0$ that it is certainly false and $P(R) = 0.5$ that we have no knowledge about it whatsoever. If on the other hand we only consider the *degree of truth* of this statement, $T(R) = 1$ means that the objects are most similar, $T(R) = 0$ that they are not similar at all and $T(R) = 0.5$ that the objects are a bit similar. Degrees of truth are oftentimes handled in fuzzy logic frameworks whereas probabilities are handled in probabilistic frameworks.

In [13], the author proposes a theoretical approach on how to combine the *probabilities* of a proposition R and *degrees of truth* of R to *degrees of belief* of R as the expected value of its degree of truth. The *degree of belief* in a proposition R is defined in this context as the strength of tendency for an agent to act as if R. Although we believe this to be a reasonable theoretical approach, the author does not give any methods on how to model dependency between variables and then perform complex inference tasks in this framework. The *Probabilistic Similarity Logic*, presented in [14], is a practical approach to incorporate degrees of truth into a probabilistic framework, with special features for set semantics relevant in the context of similarities, although it does not allow probabilistic (soft) evidence. We did not evaluate this approach on our datasets as the authors did not release their implementation yet, but we see it as a promising method to incorporate in future work.

In the following, we therefore will treat all of our continuous features as probabilities for boolean variables and also consider degrees of belief solely as probabilities. We want to use continuous features to model similarities between objects, which intuitively correspond to degrees of truth (two objects are similar to a certain degree). We are aware that we therefore approximate degrees of truth (our similarity values) by probabilities, but we believe that in our models the potential error induced by this is low enough for them to be still useful in practice.

4.2 Soft Evidence

Assume we have random variables X_e for which the evidence e (our observations) may be uncertain. Then we associate with each possible assignment $X_e = x_e$ a degree of belief $P(X_e = x_e) \in [0, 1]$. The degree of belief, which is a probability here, can be interpreted either (1) as the reliability of an observation indicating $X_e = x_e$ or (2) as the degree to which $X_e = x_e$ should be believed. The first case (1) leads to *virtual evidence* whereas the second (2) is referred to as *soft evidence*. A further discussion about the differences can be found in [15]. We will here focus on the case of soft evidence (2).

Soft evidence gives a constraint on the probability distribution over assignments to (subsets of) X_e, denoted as $P(X_e \mid e)$. For a single boolean random variable X_i with the distribution $(p_i, 1 - p_i)$ this simply requires a variable X_i to be *true* with a probability of p_i and *false* with $1 - p_i$. Several pieces of soft evidence usually influence each other and therefore cannot be treated independently without making simplifying assumptions.

4.3 Naive Bayesian Model

A naive Bayesian classifier [16, 17] is a simple probabilistic classification model. It is based on Bayes' theorem and assumes strong (naive) independence between the features. Given class labels C and a vector of features (random variables) $X = \{X_1, \ldots, X_N\}$, we assume that the value (presence/absence in the binary case) of a feature is independent of all other features, given the class. With $x = (x_1, \ldots, x_n)$ being the vector of feature values for X we can therefore compute $P(X = x \mid C = c)$ given a particular class c as

$$P(X = x \mid C = c) = \prod_{i=1}^{N} P(X_i = x_i \mid C = c) \tag{4.1}$$

To compute the class probability of a given class $C = c$, we simply apply Bayes' rule:

$$P(C = c \mid X = x) = \frac{P(X = x \mid C = c) \cdot P(C = c)}{P(X = x)} \tag{4.2}$$

Usually we compute the class probabilities for all classes in C. As the sum of all class probabilities $P(C = c \mid X = x)$ has to be one, we can see the denominator of Equation 4.2 as a normalization factor and do not have to model it explicitly. For

solving the classification task, i.e. for returning the maximum a posteriori (MAP) hypothesis c, given the feature values x, we can simply compute

$$\arg\max_{c \in C} P(X = x \mid C = c) \cdot P(C = c) \tag{4.3}$$

A naive Bayesian classifier can also be interpreted as a very simple Bayesian Network, see Figure 4.3.

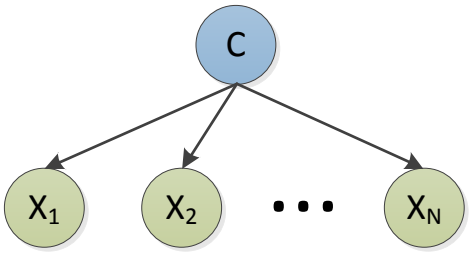


Figure 4.1: Naive Bayesian classifier as a Bayesian Network: $X_1, X_2, \ldots X_N$ are conditionally independent given the class C

When learning the parameters of a naive Bayesian classifier, we simply count the occurrences of all class- and feature-value combinations. Computing the relative frequencies of the classes and feature values in our training data allows us to make a maximum likelihood estimation for $P(C = c)$ and $P(X = x \mid C = c)$, which we usually note as (conditional) probability tables. In case some of these relative frequencies would be zero, we approximate them with a very small probability (e.g. 10^{-3}) to avoid the probability of a class c becoming zero for a feature vector x when a single feature value $x_i \in x$ did not appear in the training data for that particular class. Another method to handle this would be to make an a posteriori (MAP) estimation by applying a prior probability, e.g. by adding pseudo-counts to the relative frequencies for all feature-value/class combinations (for all entries in the conditional probability tables).

In a naive Bayesian classifier there are several different ways to treat continuous features. The authors of [18] compare some of these methods, including discretization and the approximation by a normal distribution. They conclude that no method systematically outperforms the others and propose a strategy to automatically select the best one. [16] presents an empirical study on the influence of dependencies on the error of naive Bayesian classifiers and conclude that they work best not only on completely independent features but also on functionally dependent features. We therefore present three different techniques and compare these variants in our tests.

4.3.1 Discretization

The simplest way is to discretize the continuous variables. This is done prior to learning/testing using standard discretization methods, e.g. k-means clustering of the feature values.

4.3.2 Approximation as Gaussian

A standard method to handle continuous variables is to approximate their distributions as a single Gaussian for each feature:

$$P(X = x \mid C = c) = \mathcal{N}(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4.4}$$

The maximum likelihood estimation of the distribution parameters from the training samples is straightforward. The mean μ of the Gaussian corresponds to the average value, the standard deviation σ to the standard deviation of the training samples.

A more complex model would result from the usage of kernel methods instead of a single Gaussian to approximate the distribution of the continuous features, for example the nonparametric kernel density estimation presented in [17].

4.3.3 Soft Evidence

We extended the naive Bayesian classifier to handle soft evidence in inference and learning by applying soft evidential updates (analogous to [15]) to compute posterior probabilities.

Assume we split our set of random variables X into two disjunct subsets $X = X_h \cup X_s$ with X_h containing the variables with given hard evidence X_h and the variables with given soft evidence X_s, and e being our vector of evidence values. For hard evidence, e contains the values of x_h, for soft evidence, e contains the probability for each discrete value $x_i \in \mathrm{dom}(X_i)$ for each variable $X_i \in X_s$.

The probability $P(X_s = x_s \mid e)$ is implicitly given through the soft evidence and the learned model. In order to compute $P(C = c \mid e)$, we could use a sampling based method like Gibbs sampling or MCSAT with extensions to handle soft evidence (see [15]). Instead we make an approximation here and assume the independence of our

pieces of soft evidence. We therefore can write the probability defined by our soft evidence for a particular vector of values x_s as

$$P(X_s = x_s \mid e) = \prod_{X_i \in X_s} P(X_i = x_i \mid e_i) \tag{4.5}$$

In the case of the naive Bayesian classifier this independence assumption yields a computation that is similar to using virtual evidence.

We can then calculate the class probability for a class c given our evidence e as the sum of the conditional probabilities $P(C \mid X_h, X_s)$ for all values in the domain of X_s, weighted by the given soft evidence:

$$\begin{aligned} P(C = c \mid e) &= P(C = c \mid X_h = x_h, e) \\ &= \sum_{x_s \in \prod_{X_i \in X_s} \mathrm{dom}(X_i)} P(C = c \mid X_h = x_h, X_s = x_s) \cdot P(X_s = x_s \mid e) \\ &= \sum_{x_s \in \prod_{X_i \in X_s} \mathrm{dom}(X_i)} \frac{P(X_h = x_h, X_s = x_s \mid C = c) \cdot P(C = c)}{P(X_h = x_h, X_s = x_s)} \cdot P(X_s = x_s \mid e) \end{aligned} \tag{4.6}$$

This is essentially an instance of Jeffrey's rule for updating beliefs with soft evidence [15].

In order to handle soft evidence during parameter learning, we use soft counts when approximating $P(X_i = x_i \mid C = c)$ with the relative frequencies by computing the sum of the soft evidence values for a variable X_i, given a class c. With T_c being the evidence values for training samples of the class c, we can then compute

$$P(X_i = x_i \mid C = c) = \frac{\sum_{e \in T_c} P(X_i = x_i \mid e_i)}{|T_c|} \tag{4.7}$$

Hard evidence can be seen as a border case of the soft evidence computation. For a hard evidence variable X_i, $P(X_i = x_i \mid e_i)$ is either 1 if the feature has the value x_i or 0 otherwise. The sum in Equation 4.7 is therefore equal to the number of times the feature value x_i appears in the training data. $P(X_i = x_i \mid C = c)$ is then the relative frequency of the value $X_i = x_i$ in samples for the class c, equal to the normal computation in the hard evidence case.

4.4 Markov Logic Networks (MLNs)

Markov Logic Networks [19, 20, 21] represent a powerful and coherent probabilistic framework for learning and inference tasks. Many other models like Bayesian Networks, Markov Networks, first-order logic, etc. can be transformed or incorporated into MLNs.

Markov Logic Networks are a concept that combines the semantics of probabilistic graphical models with first-order logic (FOL). In FOL, each formula is either true or false. The main idea of MLNs is to allow soft constraints on FOL formulas by assigning a weight w_i to each formula F_i. This allows the consideration of a distribution over possible worlds $\mathcal{X}$. A world $x \in \mathcal{X}$ may violate these soft constraints. Its probability then depends on the strength of the satisfied and violated constraints which is given through their weights. A violation of constraints with high weight results in a low probability for the corresponding world, whereas the violation of constraints with low weight or no violation of any constraints results in a higher probability for the corresponding world.

The set of possible worlds $\mathcal{X}$ is defined by the formulas in a first-order logic knowledge base and a finite set of constants C which represent the entities in the world. Each predicate in the knowledge base can be grounded with each of the constants in C. For example, given the binary predicate *similar(x,y)* and $C = \{A, B\}$, the possible groundings are: *similar(A,A)*, *similar(A,B)*, *similar(B,A)*, *similar(B,B)*. In each possible world, each grounding is either true or false. With N being the number of possible groundings, there are therefore $2^{|N|}$ possible worlds in $\mathcal{X}$. Each of the N groundings can be seen as a binary random variable in a set $X = \{X_1, \cdots, X_N\}$. Each possible world is then an assignment of binary truth values to all of these variables. A Markov Logic Network defines a probability distribution over the set of possible worlds $\mathcal{X}$, which are the possible assignments of truth values to the binary random variables $X_i \in X$. This probability distribution can be represented by a Markov Network (MN), a probabilistic graphical model, which is parameterized using the weights from the MLN.

As Markov Logic Networks can therefore be seen as templates for the construction of Markov Networks, we give a short introduction to Markov Networks in the next section. We give a formal definition of MLNs in Section 4.4.2 and explain exact and approximative inference methods in Section 4.4.3. In Section 4.4.3.3, we discuss inference with soft evidence. The following two Sections present known algorithms for parameter learning (Section 4.4.4) and our new methods that can also handle soft evidence (Section 4.4.5). We then compare the new methods against each other in Section 4.4.6 and explain some key differences on the basis of a small example in Section 4.4.7. There are also methods to learn the structure of a Markov Logic Net-

work. As we hand-model the MLN in our applications using our domain knowledge, and as our contribution withing MLNs is in parameter learning methods, we will not discuss structure learning here. More information on that topic can be found in [20] and [22].

4.4.1 Markov Networks

Markov Networks (MN) or Markov Random Fields (MRF) [19, 20, 21, 23] are graphical models that compactly represent the joint probability distribution of a set of variables $X = \{X_1, X_2, ..., X_n\}$.

A Markov Network consists of

- a *undirected graph* $G = (V = X, E)$. Every node in the graph represents a variable, every edge $\{X_i, X_j\} \in E$ a dependency between two variables. C is the set of cliques of G. We only regard maximal cliques here.

- a *set of potential functions* ϕ_k (often referred to as *factor potentials*). For each clique C_k with the variables $\{X_{i_1}, ..., X_{i_{N_k}}\}$ the function ϕ_k maps the domain of the clique C_k into the non-negative real numbers: $\phi_k : dom(X_{i_1}) \times ... \times dom(X_{i_{N_k}}) \to \mathbb{R}_0^+$

The joint probability distribution is given by

$$P(X = x) = \frac{1}{Z} \cdot \sum_k \phi_k(x_{\{k\}}) \tag{4.8}$$

The product includes all cliques C_k of G, $x_{\{k\}}$ is the state of the k^{th} clique in the possible world x.

Z is a normalization constant. Let $\mathcal{X}$ be the set of all possible worlds. Then

$$Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}}) \tag{4.9}$$

The set of potential functions ϕ_k maps every possible world $x \in \mathcal{X}$ to a value $v_x = \prod_k \phi_k(x_{\{k\}}) \in \mathbb{R}_0^+$. This value must be considered in relation to the values of the other possible worlds. E.g. $v_x = 10, v_{x'} = 5$ means that the world x is twice as probable as x'.

Another widely used alternative representation of a Markov Network is the log-linear model. Under the assumption that $P(X = x) > 0 \ \forall x$, the joint probability distribution can be equivalently represented as

$$P(X = x) = \frac{1}{Z} \cdot \ exp\left(\sum_i w_i \cdot f_i(x) \right) \tag{4.10}$$

An exponential sum of *weighted features* is assigned to every possible world:

- f_i: feature function of the i^{th} feature, $f_i : \times_k \ dom(X_k) \rightarrow \{0, 1\}$
- w_i: weight of the i^{th} feature, $w_i \in \mathbb{R}$

The representation is equivalent to the one describe before if there is a feature f_i for every possible state $x_{\{k\}}$ of every clique C_k, associated with the weight $w_i = log(\phi_k(x_{\{k\}}))$ (if $\phi_k(x_{\{k\}})$ is always > 0).

4.4.2 Definition and Semantics

A Markov Logic Network is defined as a set of pairs (F_i, w_i) where F_i is a first-order logic formula and $w_i \in \mathbb{R}$ is the associated real-valued weight. It represents the strength of the constraint that is represented by F_i. Together with a finite set of constants C, a MLN L defines a ground Markov Network $M_{L,C} = (X, G)$ (using the log-linear representation) as follows:

- X is a set of boolean variables. It contains one boolean variable for each grounding of each predicate in L. The set of possible worlds $\mathcal{X}$ is therefore defined as $\mathcal{X} = \mathbb{B}^{\ |X|}$.
- G is a set of weighted ground formulas, given as pairs $(\hat{F}_j, \hat{w}_j)$ It contains one such pair for each grounding $\hat{F}_j$ of each formula F_i in L with the weight $\hat{w}_j = w_i$. We define a feature $\hat{f}_j : \mathcal{X} \rightarrow \{0, 1\}$ for each such pair. $\hat{f}_j(x)$ is 1 if $\hat{F}_j$ is satisfied in the world x and 0 otherwise.

The probability distribution over all possible worlds is given by Equation 4.11, with Z being a normalization constant. $n_i(x)$ is the number of true groundings of the formula F_i in the world x.

$$\begin{aligned} P(X = x) &= \frac{1}{Z} \exp\left(\sum_j \hat{w}_j \hat{f}_j(x) \right) \\ &= \frac{1}{Z} \exp\left(\sum_i w_i n_i(x) \right) = \frac{\exp\left(\sum_i w_i n_i(x) \right)}{\sum_{x' \in \mathcal{X}} \exp\left(\sum_i w_i n_i(x') \right)} \end{aligned} \tag{4.11}$$

If a formula F_i has infinite weight w_i, it is deterministic and is therefore treated as being a hard constraint, as in first-order logic.

4.4.3 Inference

Given a Markov Logic Network L and a set of constants C, we want to compute the probability of arbitrary formulas. In addition we can have an optional evidence e. It contains the truth values for all evidence formulas E. Therefore we ask the following query: "What is the probability that F_1 is satisfied, given the evidence e?".

4.4.3.1 Exact Inference

We can formulate the queried probability as $P(F_1 \mid e, L, C)$, given in Equation 4.12. $\mathcal{X}_{F_1} \subseteq \mathcal{X}$ and $\mathcal{X}_e \subseteq \mathcal{X}$ denote the subsets of possible worlds which satisfy the formula F_1 and assign the truth values in e to the formulas in E respectively.

$$\begin{aligned} P(F_1 \mid e, L, C) &= P(F_1 \mid e, M_{L,C}) \\ &= \frac{P(F_1 \wedge e \mid M_{L,C})}{P(e \mid M_{L,C})} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_e} P(X = x)}{\sum_{x \in \mathcal{X}_e} P(X = x)} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_e} \exp\left(\sum_i w_i n_i(x) \right)}{\sum_{x \in \mathcal{X}_e} \exp\left(\sum_i w_i n_i(x) \right)} \end{aligned} \tag{4.12}$$

This computation requires to sum over all the possible worlds in $\mathcal{X}_e$, which is usually very large for non-trivial cases (in case e is empty, there are $|\mathcal{X}| = 2^{|X|}$ possible worlds).

4.4.3.2 Approximate Inference (MC-SAT)

As exact inference is computationally intractable for most non-trivial cases, usually approximative methods are used.

Gibbs sampling (widely used for normal Markov Networks) is very slow when our model contains strong dependencies and does not converge for deterministic dependencies. We therefore favor the MC-SAT (Markov Chain SATisfiability) [20] algorithm which uses a slice sampling technique and combines MCMC (Markov Chain Monte Carlo) with satisfiability (SAT) testing to achieve huge speedups compared to Gibbs sampling. Slice sampling uses auxiliary variables to decouple the state variables and thus allows rapid mixing.

We will give a short overview over MC-SAT in the following. First of all, two preparation steps have to be performed:

- Convert knowledge base (KB) to conjunctive normal form (CNF)
  $\Rightarrow$ set of ground clauses $\{c_k\}$
- Make all weights positive by negating formulas with negative weights
  $\Rightarrow \forall k \ \exp(w_k) > 1$

MC-SAT introduces an auxiliary variable u_k for each ground clause c_k. The main idea is to link them (vector U) to the possible worlds ($\mathcal{X}$), such that sampling from $\mathcal{X}$ can be done by drawing from uniform distributions for the u_k's.

The joint probability distribution for X and U is given by

$$P(X = x, U = u) = \frac{1}{Z} \prod_k I_{[0;\exp(f_k(x)\cdot w_k)]}(u_k) \tag{4.13}$$

where

- $I_{[a;b]}$ is the indicator function: $I_{[a;b]}(x) = 1$ if $x \in [a; b]$, else $I_{[a;b]}(x) = 0$.
- $P(u_k \mid x)$ is a uniform distribution over $[0; \exp(f_k(x) \cdot w_k)]$
- $P(x \mid u)$ is a uniform distribution over the slice $\mathcal{X}_u \subset \mathcal{X}$ with
  $\forall k.\ 0 \leq u_k \leq \exp(f_k(x) \cdot w_k)$.

Every possible world determines a set of possible values for the u_k's. If $u_k > 1$, $\mathcal{X}_u$ contains only worlds that satisfy c_k. Therefore every assignment of U implies a set

of formulas M that have to be satisfied: $M = \{c_k \mid u_k > 1\}$. To draw samples from $P(X = x)$, we can just sample $P(X = x, U = u)$ and ignore the values for u.

In every iteration of MC-SAT:

- If c_k is not satisfied by the current state $x^{(i)}$:
  draw u_k uniformly from $[0; 1]$. Therefore $u_k \leq 1 \leq e^{w_i}$ (because all $w_i > 0$)
  $\Rightarrow$ no requirement for c_k to be satisfied in the next state

- If c_k is satisfied by the current state $x^{(i)}$:
  draw u_k uniformly from $[0; e^{w_k}]$. Therefore $u_k > 1$ with probability $\frac{e^{w_k}-1}{e^{w_k}} = 1 - e^{-w_k}$
  $\Rightarrow$ requirement for c_k to be satisfied in the next state with probability $1-e^{-w_k}$

To shorten this case differentiation, we can write that u_k is drawn uniformly from $[0; e^{w_k \cdot f_k(x)}]$, because $f_k(x)$ is 1 if c_k is satisfied in x, and 0 otherwise.

MC-SAT determines, in every iteration, a random subset M of the currently satisfied clauses that must be satisfied in the next state by sampling all auxiliary variables u_k. The size of each interval $[0; e^{w_k}]$ is proportional to e^{w_k} and determines the probability that a clause c_k has to be satisfied.

The next state is then sampled uniformly from $\mathcal{X}_u = SAT(M)$, which is the set of states that satisfy M. In this step, it is guaranteed that all hard constraints and constraints given by the evidence are satisfied. In the end, $P(q \mid e) \approx \frac{c}{n}$ is returned. n is the total number of samples and $c = \prod_{i, x^{(i)} \models q} 1$ the number of samples in which $x^{(i)} \models q$.

Algorithm 1: **MC-SAT** $(clauses, weights, num\_samples)$

```
1: x^(0) ← Satisfy(hard clauses)
2: for i ← 1 to num_clauses do
3:    M ← ∅
4:    for all c_k ∈ clauses satisfied by x^(i−1) do
5:       with probability 1 − e^(−w_k) add c_k to M
6:    end for
7:    sample x^(i) ∼ U_SAT(M)
8: end for
```

SAT(M) can never be empty because it always contains at least the current state $x^{(i)}$. MC-SAT is initialized with $x^{(0)}$ found by a satisfiability solver such that all hard constraints (clauses with infinite weight) are satisfied.
$\mathcal{U}_{SAT(M)}$ is the uniform distribution over the set SAT(M). The uniform sampling

from this "slice" ($\mathcal{X}_u$) is computationally expensive. It is usually done using the *SampleSAT* algorithm which mixes WalkSAT (greedy) and simulated annealing (random) steps, which represent a trade-off between fast but highly non-uniform and slow but highly uniform sampling steps respectively. For more information on SampleSAT, see [24].

MC-SAT generates a Markov chain for sampling that converges to the desired stationary distribution. In practice, we can therefore always make a trade-off between the number of sampling steps and the accuracy of the results. More details on MC-SAT can be found in [20].

Another option for inference, which we will not discuss further here, is lifted inference which tries to utilize recurring structures in the Markov Network by lifting the inference problem to the level of first-order logic.

4.4.3.3 Inference with Soft Evidence

An extension of MC-SAT (see Section 4.4.3.2) to efficiently handle soft evidence updates is presented in [15]. It is referred to as *MC-SAT-PC (MC-SAT with posterior Probability Constraints)*. The key idea is to track the relative frequency of a soft evidence variable X_i throughout the sampling process. The soft evidence can then be enforced by adding X_i or $\neg X_i$ to the set M of constraints for the next MC-SAT sampling step if the relative frequency deviates too much from the given distribution.

4.4.4 Parameter Learning with Hard Evidence

In this and the next section, we want to explain how the weights of a Markov Logic Network can be learned from training data in the case of hard evidence and soft evidence, respectively. The training data is a set of formulas F and a training database, consisting of a set of constants C and truth values for all ground atoms that are defined by F and C. The training database thus defines a possible world. We typically make the *closed world assumption* and thus assign each ground atom that is not specified in the training database the truth value *false* to obtain a fully specified training set of ground atoms.

In the context of maximum likelihood parameter learning, the goal is to find weights $w = (w_1, \ldots, w_n)$ for the formulas in F such that the probability of the training database is maximized in the resulting MLN $M_{L,C}$.

We focus here on learning a generative model because we want to be able to ask our model all the different possible types of queries, depending on what questions and evidence we have in a particular task. For cases where the distinction between evidence and query atoms is known a priori at learning time, discriminative learning, as described in [20], would be the other option.

The maximization of the likelihood of the data given the model is usually done by maximizing the log-likelihood [20, 21], as presented in the next section. A rough but efficient approximation for this is the pseudo-log-likelihood. Further algorithms and optimizations for parameter learning, including second order methods, are described in [25].

4.4.4.1 Log-Likelihood

We want to maximize the probability $P(X = x \mid w)$ of the possible world x given by the training data. The weights w thereby define our current model during the iterative maximization process. We will leave w out for brevity here and in the following sections when defining the functions that are to be maximized for each of the learning methods. Here we therefore just write $P(x = x)$, which is given in Equation 4.14.

$$P(X = x) = \frac{\exp\left(\sum_i w_i n_i(x)\right)}{\sum_{x' \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x')\right)} = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{4.14}$$

From a computational point of view it is more convenient to maximize the log-likelihood $L(X = x) = \log P(X = x)$, which yields the same results.

$$L(X = x) = \sum_i w_i n_i(x) - \log(Z) \tag{4.15}$$

We can easily compute the gradient of $L(X = x)$ as follows:

$$\begin{aligned} \frac{\delta}{\delta w_i} L(X = x) &= n_i(x) - \sum_{x' \in \mathcal{X}} n_i(x') \cdot P(X = x') \\ &= n_i(x) - \sum_{x' \in \mathcal{X}} n_i(x') \cdot \frac{\exp\left(\sum_k w_k n_k(x')\right)}{\sum_{x'' \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x'')\right)} \end{aligned} \tag{4.16}$$

We can then use a standard gradient ascent method like the L-BFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno) algorithm to conveniently calculate $\arg\max_w L(X = x \mid w)$ using the gradient. In practice, however, this method has

only limited use as counting all true groundings of a formula in a database is intractable in all but the smallest domains (#P complete, see proof in [19]). We could approximate this by applying uniform sampling to obtain the relative frequency of true groundings and compute the counts using the total number of groundings, which is known. Nevertheless we would still have to perform inference over the model to calculate $P(X = x')$, which is itself also #P-complete.

4.4.4.2 Pseudo-Log-Likelihood

One way to address this problem is to approximate the log-likelihood by using the pseudo-likelihood:

$$P^*(X = x) = \prod_{k=1}^{N} P(X_k = x_k \mid \mathrm{MB}_x(X_k)) \tag{4.17}$$

where $X_k \in X$ is a ground atom, x_k is X_k's truth value in the world x, and $\mathrm{MB}_x(X_k)$ is the Markov blanket of X_k in x. The pseudo-likelihood $P^*(X = x)$ is therefore the product of the conditional likelihoods of all variables, each given the values of its direct neighbors in the network.

As before, we maximize the pseudo-log-likelihood instead, which is given by Equation 4.18. $\overline{n}_{i,k}(x)$ denotes the number of true groundings of the formula F_i in a modified world x where the truth value of the k-th ground atom, X_k, has been inverted. F_{X_k} denotes the set of indices of formulas that contain X_k.

$$\begin{aligned}
L_{PLL}(X = x) &= \log \prod_{k=1}^{N} P(X_k = x_k \mid MB_x(X_k)) \\
&= \sum_{k=1}^{N} \log \left(\frac{\exp\left(\sum_{i \in F_{X_k}} w_i n_i(x)\right)}{\exp\left(\sum_{i \in F_{X_k}} w_i n_i(x)\right) + \exp\left(\sum_{i \in F_{X_k}} w_i \overline{n}_{i,k}(x)\right)} \right) \\
&= \sum_{k=1}^{N} \log \left(1 + \exp\left(\sum_{i \in F_{X_k}} w_i \overline{n}_{i,k}(x) - \sum_{i \in F_{X_k}} w_i n_i(x) \right) \right)^{-1} \\
&= \sum_{k=1}^{N} - \log \left(1 + \exp\left(\sum_{i \in F_{X_k}} w_i (\overline{n}_{i,k}(x) - n_i(x)) \right) \right)
\end{aligned} \tag{4.18}$$

We can again optimize this by using L-BFGS. Therefore we need the gradient which is given by Equation 4.19

$$\begin{aligned}\frac{\delta}{\delta w_i} L_{PLL}(X = x) &= \sum_{k=1}^{N} [\ n_i(x) - P(X_k = x_k \mid \mathrm{MB}_x(X_k)) \cdot n_i(x) \\ &\qquad - P(X_k = \neg x_k \mid \mathrm{MB}_x(X_k)) \cdot \overline{n}_{i,k}(x)\ ] \end{aligned} \tag{4.19}$$

$\frac{\delta}{\delta w_i} L_{PLL}(X = x)$ is therefore the difference between the number of the true groundings for F_i in the training data and the expected number.

From Equation 4.18 we know that

$$P(X_k = x_k \mid \mathrm{MB}_x(X_k)) = \left(1 + \exp\left(\sum_{i \in F_{X_k}} w_i(\overline{n}_{i,k}(x) - n_i(x))\right)\right)^{-1}$$

With $p_k =_{\mathrm{def}} P(X_k = x_k \mid \mathrm{MB}_x(X_k))$ we can rewrite Equation 4.19 as follows:

$$\begin{aligned}&\frac{\delta}{\delta w_i} \log P^*(X = x) \\ &= \sum_{k=1}^{N} [\ n_i(x) - p_k \cdot n_i(x) - (1 - p_k) \cdot \overline{n}_{i,k}(x)\ ] \\ &= \sum_{k=1}^{N} (\overline{n}_{i,k}(x) - n_i(x)) \cdot (p_k - 1) \\ &= \sum_{k=1}^{N} (\overline{n}_{i,k}(x) - n_i(x)) \cdot \left(\frac{1}{1 + \exp\left(\sum_{j \in F_{X_k}} w_j \cdot (\overline{n}_{j,k}(x) - n_j(x))\right)} - 1\right)\end{aligned} \tag{4.20}$$

For the computation of the gradient it is therefore sufficient to compute the counts of true groundings $n_j(x)$ and $\overline{n}_{j,k}(x)$ for each formula in the training data, with the truth value of the k-th ground atom being maintained and being flipped, respectively. For each formula F_i we therefore only need to consider predicates that appear in the formula. Additionally, formulas whose truth value do not change by flipping the truth value of a single ground atom can be ignored as their counts cancel out when taking the difference. As all of the counts do not change during optimization, the counts and their differences only need to be computed once.

As the computation of L_{PLL} and $\frac{\delta}{\delta w_i} L_{PLL}$, Equation 4.18 and Equation 4.19 respectively, does not require inference over the model, maximization can be done very efficiently using standard optimization algorithms like L-BFGS.

The drawback of using the pseudo-log-likelihood is that it is a rough approximation that is not guaranteed to be close to the real likelihood values, especially for complex networks. In practice it can give poor results, e.g. when long chains of inference are required at query time [25].

PLL is also often subject to overfitting and thereby learning too large (hard) weights. A standard method to avoid this is to add a Gaussian prior with zero mean to the weights, though it can be problematic to find a reasonable problem-specific value for the standard deviation of the Gaussian.

4.4.4.3 Computation of Formula Frequencies

In some cases we want to model the marginal distribution over ground formulas, usually ground atoms, and to extract it directly from the evidence given in the training database. Each marginal distribution is defined over a set of formulas S_F. It either contains two formulas, F_i and its negation $\neg F_i$ or several mutually exclusive formulas. That means for each vector of constants used for grounding these formulas, there is exactly one grounded formula which is true in the evidence.

We can then use the relative frequencies of the true groundings of the formulas in the training data to directly calculate their weights. It is necessary to include all of the mutually exclusive formulas in the MLN to assign weights to them. For each formula $F_i \in S_F$, the weight can be calculated as $w_i = \log(\frac{n_i}{c})$, with n_i being the number of true groundings of F_i and c being the total number of groundings for all formulas in S_F.

This guarantees that the weights for the formulas $F_i \in S_F$ represent their marginal distribution. The weights for all other formulas, which have an intersecting set of variables with the variables used in the formulas F_i, will be learned accordingly. This method is also more efficient as the precalculated weights for the formulas in S_F are fixed and can therefore be excluded from the optimization.

In the simple boolean case, we would just have two formulas, F_1 and $F_2 = \neg F_1$. We then get $w_1 = \log(\frac{n_1}{c})$ and $w_2 = \log(\frac{n_2}{c}) = \log(1 - \frac{n_1}{c})$ with $c = n_1 + n_2$.

4.4.5 Parameter Learning with Soft Evidence

In this section we present the new methods we developed for learning MLNs with training data that contains soft evidence. These methods can be seen as extensions

or generalizations of the hard evidence learning methods presented in the previous section. In Section 4.4.6, we give an overview and a feature comparison over the new methods, and in Section 4.4.7, we discuss their applicability considering a small exemplary Markov Logic Network.

For parameter learning we make the closed world assumption and therefore assume to have evidence specifying the truth value for each ground atom. Hard evidence e, for a given model and set of constants C, thus describes a single world x with $P(X = x) = 1$. Soft evidence instead defines a probability distribution $P(X = x \mid e)$ over possible worlds $x \in \mathcal{X}$. Hard evidence can be seen as a border case of soft evidence with $\forall X_i \in X : \ P(X_i = true \mid e) \in \{0, 1\}$.

4.4.5.1 Log-Likelihood with Weighting of Formulas (LL-ISE)

A first idea is to maximize the log-likelihood (LL) of the worlds that correspond to the soft evidence, similar to the log-likelihood learning for hard evidence, presented in Section 4.4.4.1. A problem is that the soft evidence does not define a single world but a distribution over possible worlds. To be able to use a single world instead of the distribution, we work with a "soft world" in which each formula is true to a certain degree that corresponds to the probability of the given soft evidence. We therefore assign to each formula F_i its degree of belief that corresponds to the probability that a world in the evidence satisfies F_i.

This simplification serves us as an approximation of the distribution over worlds. To be able to compute this soft weight for each formula, we assume the independence of the pieces of the soft evidence (ISE assumption). We also assume that all soft evidence values are given for ground atoms.

We calculate the probability of the evidence world as

$$P_{LL\text{-}ISE}(X = x) = \frac{\exp\left(\sum_i w_i \tilde{n}_i(x)\right)}{\sum_{x' \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x')\right)} = \frac{1}{Z} \exp\left(\sum_i w_i \tilde{n}_i(x)\right) \tag{4.21}$$

with $\tilde{n}_i(x) = \sum_j \tilde{f}_j(x)$ being the soft count for the formula F_i with the groundings $\hat{F}_j$. It is similar to the number of true groundings in the hard evidence case. The only difference is that the feature $\tilde{f}_j(x)$ is real-valued instead of discrete, that is: $\tilde{f}_j : \mathcal{X} \to [0; 1]$. We calculate a value for $\tilde{f}_j(x)$ as the probability that the grounded formula $\hat{F}_j$ is satisfied in x. Let e be the evidence, our training data. Making the independent soft evidence (ISE) assumption, we can then compute $\tilde{f}_j(x) = P(\hat{F}_j \mid e)$ as follows:

1. Convert $\hat{F}_j$ into conjunctive normal form (CNF), let the disjunctions be $D_{j,k}$: $\hat{F}_j = \bigwedge_k D_{j,k}$

2. Assume independence of disjunctions: $P(\hat{F}_j \mid e) = P(\bigwedge_k D_{j,k}) = \prod_k P(D_{j,k})$

3. Calculate probabilities for the disjunctions of the ground literals $L_{j,k,l}$ under the assumption of the independence of the soft evidence variables (ISE): $D_{j,k} = \bigvee_l L_{j,k,l} = 1 - P(\bigwedge_l \neg L_{j,k,l}) = 1 - \prod_l (1 - P(L_{j,k,l}))$

4. Put it all together: $\tilde{f}_j(x) = \prod_k (1 - \prod_l (1 - P(L_{j,k,l})))$

5. $\forall j, k, l$: either $P(L_{j,k,l})$ or $P(\neg L_{j,k,l}) = 1 - P(L_{j,k,l})$ is given in the evidence e, as all soft evidence is given for ground atoms and we are making the closed world assumption for MLN learning.

As a small example, consider the formulas $f_1 = a(object) \wedge b(object)$ and $f_2 = a(object) \vee b(object)$. If we assume the given soft evidence $P(a(O1)) = 0.7$ and $P(b(O1)) = 0.4$ then the values $\tilde{n}_i(x)$ for a world x in which both $a(O1)$ and $b(O1)$ are true will be calculated as follows:

- $\tilde{n}_1(x) = P(a(O1)) \cdot P(b(O1)) = 0.7 \cdot 0.4 = 0.28$

- $\tilde{n}_2(x) = 1 - ((1 - P(a(O1))) \cdot (1 - P(b(O1)))) = 1 - (0.3 \cdot 0.6) = 0.82$

The values for all four possible worlds are:

	a, b	$a, \neg b$	$\neg a, b$	$\neg a, \neg b$
$n_1(x)$	0.28	0.42	0.12	0.18
$n_2(x)$	0.82	0.88	0.58	0.72

In case E contains only hard evidence, $L_{j,k,l}$ is either 1 (true) or 0 (false). Therefore $\tilde{f}_j(x)$ is 1 if $\hat{F}_i$ is satisfied in x and 0 if it is not satisfied. This is exactly the same definition as $\hat{f}_j(x) : \mathcal{X} \to \{0, 1\}$ for hard evidence log-likelihood (LL) learning. It follows that in this case $\tilde{n}_i(x) = n_i(x)$ and therefore $P_{LL\text{-}ISE}(X = x) = P(X = x)$. LL-ISE is therefore an extension to normal log-likelihood learning (Section 4.4.4.1), as it behaves the same in the border case of hard evidence.

For weight learning, we again maximize the log-likelihood

$$L_{LL\text{-}ISE}(X = x) = \sum_i w_i \tilde{n}_i(x) - \log(Z) \tag{4.22}$$

The gradient

$$\begin{aligned}\frac{\delta}{\delta w_i} L_{LL\text{-}ISE}(X = x) &= \tilde{n}_i(x) - \sum_{x' \in \mathcal{X}} n_i(x') \cdot P(X = x') \\ &= \tilde{n}_i(x) - \sum_{x' \in \mathcal{X}} n_i(x') \cdot \frac{\exp\left(\sum_k w_k n_k(x')\right)}{\sum_{x'' \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x'')\right)} \end{aligned} \quad (4.23)$$

Using the gradient, given by Equation 4.23, $L_{LL\text{-}ISE}$ can be maximized conveniently with a standard optimization algorithm like L-BFGS.

As the computation of Z is the same as for LL, it also requires to sum over all possible worlds. As the number of possible worlds is $|\mathcal{X}| = 2^{|X|}$, with $|X|$ being the set of ground atoms, it is computationally intractable except for the smallest MLNs and domains.

4.4.5.2 Pseudo-Log-Likelihood with Weighting of Formulas (PLL-ISE)

We define the pseudo-log-likelihood $L_{PLL\text{-}ISE}(X = x)$ for soft evidence as an extension to the pseudo-log-likelihood for hard evidence, see Section 4.4.4.2. As in LL-ISE (Section 4.4.5.1), PLL-ISE uses the soft counts $\tilde{n}_i(x)$ to approximate the counts of satisfied ground formulas in the distribution of worlds given by the soft evidence. Accordingly, it uses $\tilde{\overline{n}}_{i,k}(x)$ as the soft count for a grounded formula in which the k-th ground atom X_k has been inverted. In the computation of $\tilde{\overline{f}}_i(x)$ for $\tilde{\overline{n}}_{i,k}(x)$, the inverted ground atom receives a probability $P(\neg X_k) = 1 - P(X_k)$, with $P(X_k)$ being the soft evidence value for the ground atom X_k. All soft evidence is given at the level of ground atoms as we also make the independent soft evidence (ISE) assumption for PLL-ISE. This yields Equation 4.24 and Equation 4.25 for the pseudo-log-likelihood and its gradient respectively.

$$L_{PLL\text{-}ISE}(X = x) = \sum_{k=1}^{N} -\log\left(1 + \exp\left(\sum_{i \in F_{X_k}} w_i \cdot (\tilde{\overline{n}}_{i,k}(x) - \tilde{n}_i(x))\right)\right) \quad (4.24)$$

$$\begin{aligned}&\frac{\delta}{\delta w_i} L_{PLL\text{-}ISE}(X = x) \\ &= \sum_{k=1}^{N} \left(\tilde{\overline{n}}_{i,k}(x) - \tilde{n}_i(x)\right) \cdot \left(\frac{1}{1 + \exp\left(\sum_{j \in F_{X_k}} w_j \cdot (\tilde{\overline{n}}_{j,k}(x) - \tilde{n}_j(x))\right)} - 1\right)\end{aligned} \quad (4.25)$$

As the pseudo-log-likelihood is a rough approximation, its computation is very fast compared to the other methods, but it often gives poor results in non-trivial scenarios, similar to the hard evidence PLL (Section 4.4.4.2). In combination with precalculated weights (see Section 4.4.4.3), we experienced it failing completely by optimizing toward zero weights.

4.4.5.3 Log-Likelihood with Sampling and Weighting of Formulas (SLL-ISE)

As the computational problem with LL-ISE is the calculation of the normalization constant Z, we propose to use a sampling based approach to calculate an approximation $\tilde{Z}$ for it. With S_u being a multiset of uniformly sampled worlds, the log-likelihood can be approximated as

$$\begin{aligned} L_{SLL\text{-}ISE}(X = x) &= \sum_i w_i \tilde{n}_i(x) - \log(\tilde{Z}) \\ &= \sum_i w_i \tilde{n}_i(x) - \log\left(C \cdot \sum_{s \in S_u} \exp\left(\sum_k w_k n_k(s)\right)\right) \end{aligned} \tag{4.26}$$

with $C = \frac{|\mathcal{X}|}{|S_u|}$. If we draw the same number of samples $|S|$ in each step, C does not matter for the optimization and can be left out.

But there is a problem with sampling the worlds in S_u uniformly. Given a large number of ground atoms and non-zero weights w, it is not practicable, because usually only few worlds have a significant high probability whereas the probability of the other worlds is close to zero. If we apply uniform sampling, the number of samples would have to be extremely high to get, with a high enough probability, a sufficient number of samples where $P(X = s) \propto \exp\left(\sum_k w_k n_k(s)\right) \gg 0$. Otherwise, if $\tilde{Z}$ in Equation 4.26 is approximately zero, the optimization would fail.

Instead we run MC-SAT (Section 4.4.3.2) to sample from the prior using the current weights w, but not the training evidence e. We take the worlds from the Markov chain generated by MC-SAT as samples and denote the multiset of sampled worlds as S. The problem with that approach is that by running MC-SAT, the worlds in S are sampled with the probability $P(X = s)$. It is therefore possible that multiple samples s_i contribute to the normalization Z for the same world x. Assume that we would group the worlds in S into disjoint subsets S_x of similar samples for each world x and that we draw a sufficient number of samples. Then with $|S_x|$ being the number of samples that represent world x:

$$\forall x \in \mathcal{X} : \ \frac{|S_x|}{|S|} \approx P(X = x) \propto \exp\left(\sum_k w_k n_k(x_k)\right) \tag{4.27}$$

This means that with $m \cdot P(X = x)$ being approximately a multiple of the relative frequency with which we sample x:

$$\begin{aligned} \sum_{s \in S} \exp\left(\sum_k w_k n_k(s)\right) &\approx \sum_{x \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x)\right) \cdot m \cdot P(X = x) \\ &\propto \sum_{x \in \mathcal{X}} \left(\exp\left(\sum_k w_k n_k(x)\right)\right)^2 \end{aligned} \tag{4.28}$$

which is not proportional to $\tilde{Z}$.

We therefore propose to remove duplicates from the set of samples S, which we denote $\bar{S}$. In the border case of a high enough (or infinite) number of samples and $\forall x \in \mathcal{X} : P(X = x) > 0$, it contains a sample for each world, which means $\bar{S} = \mathcal{X}$. This would give us the same computation as LL-ISE. In the usual case, where we have a limited number of samples $\ll |\mathcal{X}|$, it can still be used for a good approximation of Z, as we use the samples to sum up values that are proportional to $P(X = s)$. With a high probability we will therefore add samples to $\bar{S}$ which have a large contribution to $\tilde{Z}$ and leave out samples that would have a low contribution to it. We therefore need a much smaller number of samples to get a useful approximation as if we would use uniform sampling. With

$$\tilde{Z} \approx \bar{C} \cdot \sum_{s \in \bar{S}} \exp\left(\sum_k w_k n_k(s)\right) \tag{4.29}$$

and $\bar{C} = \frac{|\mathcal{X}|}{|\bar{S}|}$ we can rewrite Equation 4.26 as

$$L_{SLL\text{-}ISE}(X = x) \approx \sum_i w_i \tilde{n}_i(x) - \log\left(\bar{C} \cdot \sum_{s \in \bar{S}} \exp\left(\sum_k w_k n_k(s)\right)\right) \tag{4.30}$$

A further aspect to consider is whether we should add the summand of our soft evidence world $\sum_i w_i \tilde{n}_i(x)$ to the normalization $\tilde{Z}$. On the one hand it would ensure to keep the approximated log-likelihood below 0, which is a nice property if we try to interpret the likelihood, but not necessary for the optimization. On the other hand, we thereby add a mixture of worlds to the normalization that might already be (partially) given in the samples. Contrary to first intuition, mathematically it does not matter. All we want to do is to find the $\arg\max_w$ of the function $\frac{\exp(\sum_i w_i \tilde{n}_i(x))}{\tilde{Z}}$. With $a(w)$ being the term corresponding to our evidence world and $b(w)$ being the term corresponding to all other worlds, we can write it as $\frac{a(w)}{a(w)+b(w)}$ or $\frac{a(w)}{b(w)}$ in case we add the evidence world to $\tilde{Z}$ and in case we do not, respectively. As these terms are always positive, with

$$\arg\max_w \frac{1}{f(w)} = \arg\min_w f(w) \qquad \forall\ f(w) > 0 \tag{4.31}$$

we can see that

$$\arg\max_w \frac{a(w)}{a(w)+b(w)} = \arg\max_w \frac{1}{1+\frac{b(w)}{a(w)}} =$$
$$\arg\min_w 1+\frac{b(w)}{a(w)} = \arg\max_w \frac{a(w)}{b(w)} \qquad \forall\ a(w), b(w) > 0 \tag{4.32}$$

and it mathematically therefore does not matter which variant we use. But in our application, if we use floating point values with limited precision, it can be numerically more stable in some cases to leave the evidence world out. Otherwise, when we calculate: $\log(a(w)) - \log(a(w)+b(w))$ and $a(w) \gg b(w)$, we might get $a(w)+b(w) = a(w)$ due to precision loss. This would make the whole expression that is to be maximized equal to $\log(a(w)) - \log(a(w)) = 0$.

The gradient of SLL-ISE is

$$\begin{aligned}\frac{\delta}{\delta w_i} L_{SLL\text{-}ISE}(X=x) &= \tilde{n}_i(x) - \sum_{s\in S_u}\left(n_i(s)\cdot\frac{\exp\left(\sum_k w_k n_k(s)\right)}{\tilde{Z}}\right) \\ &\approx \tilde{n}_i(x) - \sum_{s\in S}\left(\frac{n_i(s)}{|S|}\right)\end{aligned} \tag{4.33}$$

We can make this approximation because the probability of the samples $s \in S$, when sampling with MC-SAT, is already $P(X=s) \approx \frac{\exp\left(\sum_k w_k n_k(s)\right)}{\tilde{Z}}$.

Similar to $L_{LL\text{-}ISE}$, $L_{SLL\text{-}ISE}$ can be maximized using L-BFGS. As we also use the soft counts $\tilde{n}_i$, we still have to make the assumption of independent soft evidence. Using MC-SAT to sample expected values for n_i is similar to the Contrastive Divergence method presented in [25] for hard evidence learning.

4.4.5.4 Log-Likelihood with Double Sampling and Weighting of Worlds (DSLL-WW)

In all methods we have presented so far, we approximate the distribution of worlds given by the soft evidence as a single soft world by assigning a degree of belief to each formula. With DSLL-WW, however, we do not make this simplification, but draw samples from the distribution of worlds defined by the training data. The evidence is therefore not seen as a single soft world anymore but as a weighted sum of worlds. This yields the advantage that we do not have to make the assumption of independent soft independence. DSLL-WW builds upon the method SLL-ISE and uses sampling both for the evidence and the normalization term.

With this method, we do not maximize the probability of a single world $P(X = x)$ anymore, but the weighted average of probabilities of worlds sampled from the distribution of worlds that is defined by the soft evidence. This is given in Equation 4.34, with S_e being the set of worlds sampled to approximate the distribution given by the training data. This sampling is done using MC-SAT under consideration of the given soft evidence e. Therefore the worlds in S_e are sampled with the probability $\tilde{P}(X = s \mid e)$ which approximates $P(X = s \mid e)$.

$$\begin{aligned}\sum_{x\in\mathcal{X}} P(X = x \mid e) \cdot P(X = x) &\approx \sum_{x\in\mathcal{X}} \tilde{P}(X = x \mid e) \cdot \tilde{P}(X = x) \\ &= \frac{1}{|S_e|} \cdot \sum_{s\in S_e} \tilde{P}(X = s) \\ &= \frac{1}{|S_e|} \cdot \sum_{s\in S_e} \frac{\exp\left(\sum_k w_k n_k(s)\right)}{\tilde{Z}} \end{aligned} \tag{4.34}$$

We again use the logarithm of this value:

$$\begin{aligned} L_{DSLL\text{-}WW}(e) &= \log\left(\frac{1}{|S_e|} \cdot \sum_{s\in S_e} \exp\left(\sum_k w_k n_k(s)\right)\right) - \log(\tilde{Z}) \\ &= \log\left(\frac{1}{|S_e|} \cdot \sum_{s\in S_e} \exp\left(\sum_k w_k n_k(s)\right)\right) - \log\left(\bar{C} \cdot \sum_{s\in \bar{S}} \exp\left(\sum_k w_k n_k(s)\right)\right) \end{aligned} \tag{4.35}$$

The normalization $\tilde{Z}$ is calculated similar to SLL-ISE, $\bar{S}$ is sampled without evidence and does not contain duplicate worlds, as described in Section 4.4.5.3.

The gradient, given in Equation 4.36, is the difference between the average count of true groundings of a formula F_i in the sampled worlds given by the evidence and in the worlds sampled without considering the evidence.

$$\frac{\delta}{\delta w_i} L_{DSLL\text{-}WW}(e) = \sum_{s\in S_e} \left(\frac{n_i(s)}{|S_e|}\right) - \sum_{s\in \bar{S}} \left(\frac{n_i(s)}{|\bar{S}|}\right) \tag{4.36}$$

4.4.5.5 Further Approaches

Another idea for a learning algorithm, derived from LL-ISE, is to not use the soft counts but to compute a value for the mixture of all possible worlds according to the given soft evidence. We weigh each world x by its probability λ_x. It is computed by multiplying the soft evidence values for each ground atom X_i as we treat them

again as being independent probabilities. Given the evidence e, we can compute it as

$$P_{LL\text{-}ISEWW}(e) = \frac{\sum_{x \in \mathcal{X}} \left(\exp\left(\sum_i w_i n_i(x)\right) \cdot \lambda_x\right)}{\sum_{x \in \mathcal{X}} \exp\left(\sum_k w_k n_k(x)\right)} \tag{4.37}$$

with

$$\lambda_x = \prod_{X_i} P(X_i \mid e) \tag{4.38}$$

The problem with this idea is that it will lead to an optimization of the weights such that they are maximizing the single world with the maximum λ_x (assuming no other worlds have the same maximum λ_x). This can be seen by transforming Equation 4.37:

$$\begin{aligned} P_{LL\text{-}ISEWW}(e) &= \frac{\sum_{x \in \mathcal{X}} \left(\exp\left(\sum_i w_i n_i(x)\right) \cdot \lambda_x\right)}{Z} \cdot \frac{\frac{1}{Z}}{\frac{1}{Z}} \\ &= \sum_{x \in \mathcal{X}} P(X = x) \cdot \lambda_x \end{aligned} \tag{4.39}$$

We just maximize the sum of the products of the probability of the soft evidence times the probability given the current weights for each world. With x^* being the most probable world according to the evidence, i.e. $\forall\ x_i :\ \lambda_{x_i} \leq \lambda_{x^*}$, maximizing Equation 4.39 leads to weights such that $P(X = x^*)$ is close to 1 because

$$1 \cdot \lambda_{x^*} \geq P(X = x^*) \cdot \lambda_{x^*} + \sum_{x \in \mathcal{X} \setminus \{x^*\}} P(X = x) \cdot \lambda_x \tag{4.40}$$

with $\sum_{x \in \mathcal{X}} \lambda_x = 1$. This means that giving a probability > 0 to any other world than x^* would reduce the sum which is to be maximized.

Although that approach does not work, it might still be a reasonable idea to use the world probability λ_x during optimization. We therefore propose to minimize the deviation of the world probabilities $P(X = x)$ given by the weights from the world probabilities λ_x given by the soft evidence. This error measure can be formulated as

$$\sum_{x \in \mathcal{X}} |P(X = x) - \lambda_x| \tag{4.41}$$

Preliminary tests in small MLNs showed reasonable results using only hard or only soft evidence, but failed in the mixed case. We see the minimization of an error measure instead of the maximization of the likelihood of a world or a mixture of worlds as an interesting topic for further research.

4.4.6 Comparison

We now want to give a comparison between the methods we presented in the previous section. The following feature matrix highlights the key differences of the different algorithms.

	LL-ISE	PLL-ISE	SLL-ISE	DSLL-WW
Assume independent soft evidence	✓	✓	✓	
Assume soft evidence world	✓	✓	✓	
Approximative		✓	✓	✓
Sampling-based			✓	✓
Runtime	−	++	+	
Applicable for complex models			(✓)	(✓)

LL-ISE is nice in theory and useful as a baseline for testing new methods, but in practice it is usually intractable to explicitly compute values for all possible worlds. PLL-ISE is computationally very efficient through simplification of the problem, but therefore fails in many relevant applications, as pseudo-log-likelihood does for hard evidence. SLL-ISE gives a good approximation of LL-ISE while still being tractable for larger models. It allows to make the trade-off between precision and the number of samples which is proportional to the computation time. We believe it therefore to be the best choice for many applications. DSLL-WW is the only method with which we do not assume a single evidence world defined through soft counts but consider the real distribution of worlds. This allows us to drop the assumption of independent soft evidence that is necessary for all the other methods. It requires more time than SLL-ISE as it uses two different sampling based approximations, but if we want to consider possible dependencies between pieces of soft evidence, then it is our only choice here.

4.4.7 Example and Discussion

We will illustrate some of the differences of the presented methods in the following small example and discuss the results. The example MLN and evidence databases are written in the syntax used by the *Progcog* MLN tools [4] which extend the syntax used by the *Alchemy* framework [26]. Our example MLN is given by the following:

```
//predicate declarations:
a(object)
b(object)
```

```
//formulas:
#fixWeightFreq
0 a(x)
#fixWeightFreq
0 !a(x)

0 a(x) ^ b(x)
0 a(x) ^ !b(x)
0 !a(x) ^ b(x)
0 !a(x) ^ !b(x)
```

The first section defines all used predicates, the second section our formulas. "!$a(x)$" is simply the notation for "not a(x)". The weights for the formulas marked with "#fixWeightFreq" are precalculated according to their relative frequencies in the training database, as explained in Section 4.4.4.3. The zeros in front of the formulas are the initial weights for the optimization process which we set as a default to 0.

We precalculate and fix the weights for $a(x)$. The other four formulas in our MLN, representing all possible conjunctions between $a, b, \neg a, \neg b$, are used to learn the conditional probability $P(b \mid a)$.

First we will train the MLN with hard evidence, second with soft evidence and thereby discuss the results. As hard evidence training data, we use the following small database using the domain (set of constants) $C = \{O1, O2, O3\}$.

```
a(O1)
a(O2)
!a(O3)

b(O1)
b(O2)
!b(O3)
```

In all of our training databases, we make the closed world assumption. That means that all unspecified ground atoms are assumed to have a truth value of *false* for training. After learning our model with log-likelihood (LL, Section 4.4.4.1), we get the following weights:

```
-0.405465    a(x)
-1.098612    !a(x)
6.434239     a(x) ^ b(x)
-7.087464    a(x) ^ !b(x)
-5.781015    !a(x) ^ b(x)
6.434239     !a(x) ^ !b(x)
```

With our testing domain $C = \{O\}$, querying for "$a(O) \wedge b(O)$" gives us a probability of 0.6667. This is exactly what we would expect, as $P(a \wedge b) = P(b \mid a) \cdot P(a) = 1 \cdot \frac{2}{3} \approx 0.6667$. The MLN learns the conditional probability $P(b|a) = 1$ because in the training data, $b(x)$ is always true if and only if $a(x)$ is also true.

Pseudo-log-likelihood (PLL, Section 4.4.4.2) learning fails in this example. If we precalculate the weights for $a(x)$, PLL just learns zero weights for the other formulas. If we refrain from doing so, the weights are

```
-1.775603    a(x)
1.775603     !a(x)
7.884686     a(x) ^ b(x)
-9.660289    a(x) ^ !b(x)
-7.926497    !a(x) ^ b(x)
9.702100     !a(x) ^ !b(x)
```

which give incorrect results. Especially the weight for $!a(x)$ is too large. Querying for "$a(O) \wedge b(O)$" gives a probability of 0.004639 which is not correct. We might be able to learn better weights by adding a Gaussian prior to the weights in order to prefer smaller values, but this would require additional problem-specific hand tuning.

Now we want to consider a training database containing soft evidence. Each line has the format "$P(F_i \mid e) \quad F_i$":

```
0.6667 a(O1)
0.6667 b(O1)
```

It is important to note that although again $a(x)$ is true in $\frac{2}{3}$ of the possible worlds and $b(x)$ is true in $\frac{2}{3}$ of the possible worlds, this data gives a fundamentally different model than the previous hard training database. It defines no dependencies between $a(x)$ and $b(x)$ whatsoever, that means, it does not say anything about whether $a(x)$ is true in the same worlds in which $b(x)$ is true or not.

Using SLL-ISE (Section 4.4.5.3), we learn the following weights:

```
-0.405415    a(x)
-1.098712    !a(x)
0.346669     a(x) ^ b(x)
-0.346588    a(x) ^ !b(x)
0.346609     !a(x) ^ b(x)
-0.346690    !a(x) ^ !b(x)
```

Inference gives us $P(a(O) \wedge b(O)) = 0.4445$. This is $P(a \wedge b) = P(a) \cdot P(b) = 0.6667 \cdot 0.6667 \approx 0.4445$, which is correct because we assume that the pieces of evidence given for $a(x)$ and $b(x)$ are independent of each other.

The approximative sampling-based methods SLL and DSLL (Section 4.4.5.3 and 4.4.5.4) lead to weights that give $P(a(O) \wedge b(O))$ a value of 0.450800 and 0.475435, respectively. We used 1000 samples for S and S_e in each iteration[1]. These probabilities are approximately correct and vary slightly among different runs as the sampling done by MC-SAT is a randomized method.

If we want to get the same model as for hard evidence, we have to specify the dependency between $a(x)$ and $b(x)$ explicitly, for example as a soft evidence for $P(a \wedge b) = 0.6667$. Our learning algorithm that can handle this is DSLL-WW. Using 10000 samples for both sample sets in each iteration, we then get for our query "$a(O) \wedge b(O)$" a probability of 0.668525 which is approximately the same as in the hard evidence case.

One general problem with these sampling based methods is that we have to guess a sufficient number of samples for each problem and the desired precision. A too large number does not hurt, but the learning may take a very long time (the runtime scales linearly with the number of samples, not taking effects on L-BFGS convergence into account). An insufficient number on the other hand leads to imprecision or in the worst case to completely wrong results as the optimization fails. For sampling $s \in S_e$ from the soft evidence distribution of worlds, one option would be to stop sampling when the deviation of the soft evidence values from relative frequency in the sampled Markov chain falls below some threshold. For the sampling of $s \in S$ for normalization, we might stop when the change of $\tilde{Z}$ throughout a certain number of samples falls below some threshold. We see defining these thresholds and finding reasonable values for them as a topic for future research.

[1] In this example, it would be sufficient to use a smaller number of samples for S as there are only four different possible worlds and duplicates get removed. That means, we could stop sampling once these four worlds are found. This is not possible in more complex examples with a higher number of ground atoms $|G|$, as $2^{|G|}$ possible worlds exist.

5 Learning Organizational Principles

Having identified the important principles in Section 2, we want to learn a model that allows us to solve the classification task of choosing the best location to place a previously unseen object in the kitchen. We additionally want to gain insights into the relevant organizational principles at each location in a kitchen by analyzing feature importance.

5.1 Features

As outlined above, we believe that organizational principles are governed by the notion of similarity. Similarity, however, can be defined in manifold ways: We could consider the similarity along any dimension, including the size, shape, weight, colour, value, fitness for a particular purpose, etc. of the object as features that would allow to identify the organizational principles that are characteristic for a specific location. Of course, we can also consider aggregates of the aforementioned similarities that consider an arbitrary number of dimensions at the same time.

In our experiments, we consider the following features, all of which correspond to principles identified in Section 2:

- *WUP similarity:* a semantic degree of similarity between concepts in an ontology (where concepts in the ontology correspond to types of objects); it gives an indication of how similar the types of objects are.

- *Purpose:* what the object can be used for, as defined through super-concepts in our ontology to which the type of the object belongs (four binary features indicating whether the object is a *FoodVessel, PhysicalDevice, FoodOrDrink* or *FoodIngredient*)

- *MealRelevance:* five binary features indicating whether the type of object is typically used for *Breakfast, PrincipalMeal, Coffeebreak, Snack* or *Sandwich*

- *Size:* discretized size of the object according to its largest dimension, *size* $\in$ {s, m, l}
- *Shape:* discrete values for the shape of the object, *shape* $\in$ {box, cylindric, flat, bag, other}

The WUP similarity is one particularly versatile similarity measure. It was originally defined by Wu and Palmer in [27] in the context of automatic translations. For two concepts in an ontology, it defines a similarity value in the interval $[0; 1]$, taking the depth of the concepts and the depth of their lowest common super-concept (LCS) into account:

$$wupSim(C_1, C_2) = \frac{depth(LCS(C_1, C_2))}{\frac{1}{2}\left(depth(C_1) + depth(C_2)\right)} \tag{5.1}$$

The reflexive case is defined as $wupSim(C, C) = 1$. The computation is illustrated in Figure 5.1, showing a simplified version of our kitchen ontology with $wupSim(\text{CoffeeCup}, \text{SodaGlass}) = 0.5$.

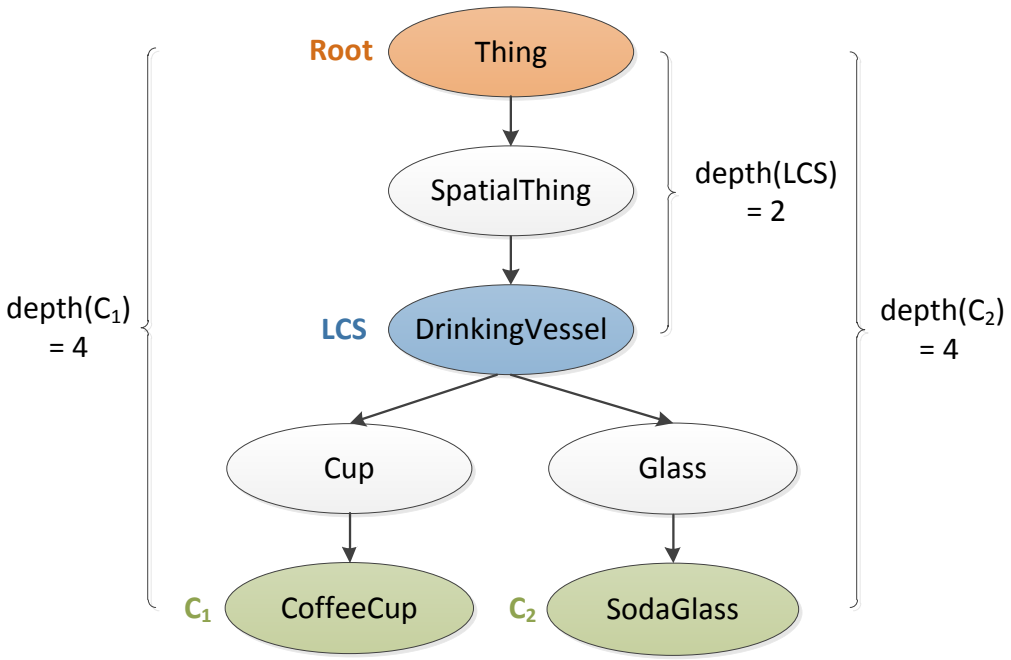


Figure 5.1: Example for path lengths used to calculate the WUP-Similarity

The WUP similarity's versatility is due to the flexible ways in which we can define concepts in our ontology. Concepts can have multiple super-concepts, creating different subtrees in the ontology that correspond to distinct aspects of a particular type of object. For example, a refrigerator can be seen as an electrical household appliance, a cooling device, or just as a box-shaped container. Depending on the situation at hand, each of these views may be more or less relevant. These different sub-trees in the ontology lead to multiple connections between two concepts, each having possibly different lengths that correspond to the semantic distance in that particular respect. The WUP similarity typically computes the minimum of all these distances because it considers the lowest common super-concept (LCS).

The layout of the ontology thus influences the computed similarity values. Note that the ontologies we used were not specifically designed for computing the similarities. We rather extended an existing ontology with classes that were automatically derived from an online shop's website. Since the distances computed from this ontology appear to be meaningful to humans, it seems to be close to a "natural ontology" of household objects.

We visualize WUP similarities in Figure 5.2, where we show an excerpt of a graph containing a small subset of the concepts from one of our real kitchen datasets. We defined a distance measure $M_{ij} := 1 - wupSim(C_i, C_j)$, calculating the distances between all pairs of concept and using multidimensional scaling to visualize the distance matrix M in two dimensions. Concepts located close together in the graph have high pairwise similarity (low distance). Each shape/color indicates a different location where objects of the corresponding concepts are stored in one of our real-world datasets. We observe that most concepts found at the same location are located in clusters separate from the others, which indicates high discriminative power of the WUP similarity for our classification task.

We refer to the group of products located at one place as a *location*. Based on the WUP similarities between pairs of objects, we define two similarity measures between a single object O and a location L as follows:

$$maxWup(O, L) = \max_{O' \in L} wupSim(class(O), class(O')) \tag{5.2}$$

$$avgWup(O, L) = \sum_{O' \in L} \frac{wupSim(class(O), class(O'))}{|L|} \tag{5.3}$$

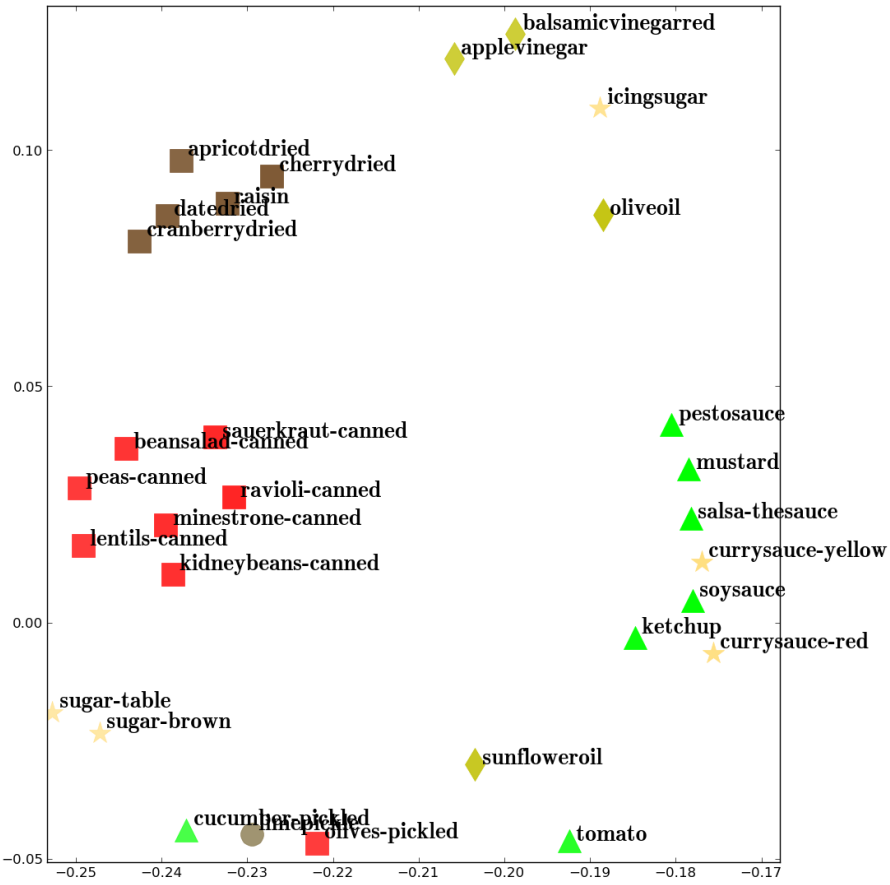


Figure 5.2: Visualization of pairwise distances between concepts, based on the WUP similarity. Each shape/color indicates a location in the kitchen.

5.2 Classifiers

We now describe the classifiers we tested for the task of allocating an object to its most appropriate storage location. For our classifiers, we use one *avgWup* and one *maxWup* feature for each location, unless noted otherwise.

5.2.1 Maximum WUP Similarity

The first set of primitive classifiers return the location with the maximum WUP similartiy between the object and the location. We define two classifiers, one using the maximum *avgWup*, the other using the maximum *maxWup* similarity. The latter

is equivalent to returning the location that contains the most similar object with respect to WUP similarity.

5.2.2 Decision Trees

We applied unpruned C4.5 decision trees with the discrete and continuous features described in section 5.1. We use the implementation from the Weka machine learning suite [28] called J48.

5.2.3 Boosted Decision Trees

We applied AdaBoost (Weka implementation called AdaBoostM1) with pruned C4.5 decision trees (J48), using the Weka default parameters.

5.2.4 Support Vector Machines (SVM)

We used the Weka SVM implementation called SMO with polynomial kernels and the default parameters ($C = 1$).

5.2.5 Naive Bayes

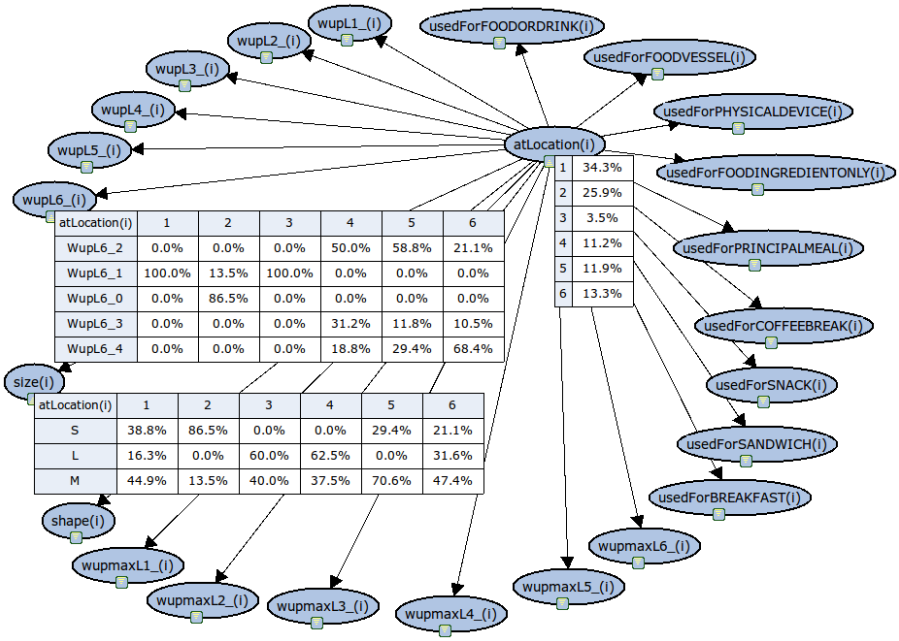
In a naive Bayesian classifier, the WUP similarities, as continuous features, can be treated in various ways, as discussed in Section 4.3. We therefore compare three variants in our tests:

- *NB discrete:* We use k-means clustering to discretize each of the continuous features with $k = 5$. Figure 5.3 shows the network structure and some exemplary (conditional) probability tables of one instance of the resulting classifier.

- *NB continuous:* We approximate the distribution of each continuous feature with a Gaussian, increasing the variance by 0.05 to avoid overfitting.

- *NB soft:* We treat the degree of similarity as a degree of belief, using a boolean variable for each continuous feature and applying soft evidential updates to compute posterior probabilities, as described in Section 4.3.3. As we did not

yet implement the described soft evidence inference algorithm for our naive Bayes classifier, we exported the trained classifier as a Markov Logic Network with the same semantics (see [21] for details on the conversion of Bayesian Networks into MLNs) and performed exact inference with soft evidence (see Section 4.4.3.3) in the MLN.



atLocation(i)	1	2	3	4	5	6
WupL6_2	0.0%	0.0%	0.0%	50.0%	58.8%	21.1%
WupL6_1	100.0%	13.5%	100.0%	0.0%	0.0%	0.0%
WupL6_0	0.0%	86.5%	0.0%	0.0%	0.0%	0.0%
WupL6_3	0.0%	0.0%	0.0%	31.2%	11.8%	10.5%
WupL6_4	0.0%	0.0%	0.0%	18.8%	29.4%	68.4%

1	34.3%
2	25.9%
3	3.5%
4	11.2%
5	11.9%
6	13.3%

atLocation(i)	1	2	3	4	5	6
S	38.8%	86.5%	0.0%	0.0%	29.4%	21.1%
L	16.3%	0.0%	60.0%	62.5%	0.0%	31.6%
M	44.9%	13.5%	40.0%	37.5%	70.6%	47.4%

Figure 5.3: Graphical representation of a discrete naive Bayes model for our mockup kitchen data set with three exemplary (conditional) probability tables

When interpreting similarity values as degrees of belief, we intuitively want to have the minimum similarity correspond to "not similar", the maximum similarity to "similar" and values in between to "similar to some degree". Most WUP similarities in our dataset are approximately between 0.4 and 0.8, therefore normalization is desirable. We normalize the WUP similarity features for the soft and continuous naive Bayesian classifiers to the $[0; 1]$ interval by applying a linear scaling on the values occurring in the training data. Values in the test data are scaled in the same way and clipped to the $[0; 1]$ interval in case they exceed its limits.

5.2.6 Markov Logic Networks

In the following sections, we present, in order of increasing complexity, several models based on Markov Logic Networks (see Section 4.4). They all use soft evidence during parameter learning and inference to handle our continuous WUP similarity features, analogous to the soft naive Bayes classifier presented in the previous

section. In Section 5.2.6.5 we discuss the challenges we faced when trying to apply these models in practice.

For brevity, we include only one *wupSimilarity* predicate that represents our *maxWup* (or *avgWup*) feature. Further features, assuming their independence, can easily be added by including predicates and formulas similar to the ones containing *wupSimilarity*.

5.2.6.1 MLN Using Similarity to One Location

Our first MLN has two predicates, *atLocation(O, L)* stating that the object O is located at the location L and *wupSimilarity(O, L)* stating that O is similar to the objects at L. For weight learning, we use hard evidence for the locations of the known objects and state the continuous (normalized) WUP similarities by giving soft evidence for the *wupSimilarity(O, L)* predicate.

Our simplest MLN just models the prior probabilities at each location L and the conditional probabilities $P(\textit{wupSimilarity(O, L)} \mid \textit{atLocation}(O, L))$. We define it as follows:

```
atLocation(object, location!)
wupSimilarity(object, location)

#fixWeightFreq
0 atLocation(o, L1)
#fixWeightFreq
0 !atLocation(o, L1)

0 atLocation(o, L1) ^ wupSimilarity(o, L1)
0 atLocation(o, L1) ^ !wupSimilarity(o, L1)
0 !atLocation(o, L1) ^ wupSimilarity(o, L1)
0 !atLocation(o, L1) ^ !wupSimilarity(o, L1)

[...]

#fixWeightFreq
0 atLocation(o, L6)
#fixWeightFreq
0 !atLocation(o, L6)

0 atLocation(o, L6) ^ wupSimilarity(o, L6)
0 atLocation(o, L6) ^ !wupSimilarity(o, L6)
0 !atLocation(o, L6) ^ wupSimilarity(o, L6)
```

```
0 !atLocation(o, L6) ^ !wupSimilarity(o, L6)
```

As this model does not include any formulas that can model a dependency between a location L_i and the WUP similarity to another location L_j with $L_i \neq L_j$, it is less powerful than the naive Bayes classifier model presented in Section 5.2.5, which uses the WUP similarities to all locations as features.

5.2.6.2 MLN Using Similarity to All Locations

We now want to include formulas to model these dependencies by including formulas expressing the conditional probabilities $P(wupSimilarity(O, L_i) \mid atLocation(O, L_j))$ for all pairs of i and j. In the following MLN, this is denoted as $+l$ and $+l2$, which are automatically expanded for all values of the domain of *location*.

```
location = {L1, L2, L3, L4, L5, L6}

atLocation(object, location!)
wupSimilarity(object, location)

0 atLocation(o, +l) ^ wupSimilarity(o, +l2)
0 atLocation(o, +l) ^ !wupSimilarity(o, +l2)
0 !atLocation(o, +l) ^ wupSimilarity(o, +l2)
0 !atLocation(o, +l) ^ !wupSimilarity(o, +l2)

#fixWeightFreq
0 atLocation(o, L1)
#fixWeightFreq
0 !atLocation(o, L1)

[...]

#fixWeightFreq
0 atLocation(o, L6)
#fixWeightFreq
0 !atLocation(o, L6)
```

This MLN would have the same expressive power as our soft naive Bayes classifier in case we use the same set of features. Compared to the previous, less complex, model, this implies a higher computational effort, as this MLN contains $4|\mathcal{L}|^2+2|\mathcal{L}|$ formulas whereas the previous one contained only $6|\mathcal{L}|$.

5.2.6.3 MLN Using Pairwise Similarities between Objects

Our next step is to use the power of MLNs to model dependencies that we cannot capture with our other classifiers. Instead of modeling dependencies between objects and locations using the *maxWup* and *avgWup* aggregations, we directly include the pairwise WUP similarities between objects into our model, expressed in the following MLN by the predicate *wupSimilarity(o1, o2)*.

```
atLocation(object, location!)
wupSimilarity(object, object)

#fixWeightFreq
0 atLocation(o, L1)
#fixWeightFreq
0 !atLocation(o, L1)

0 wupSimilarity(o1, o2) ^ atLocation(o1, L1) ^ atLocation(o2, L1)
0 wupSimilarity(o1, o2) ^ atLocation(o1, L1) ^ !atLocation(o2, L1)
0 wupSimilarity(o1, o2) ^ !atLocation(o1, L1) ^ atLocation(o2, L1)
0 wupSimilarity(o1, o2) ^ !atLocation(o1, L1) ^ !atLocation(o2, L1)
0 !wupSimilarity(o1, o2) ^ atLocation(o1, L1) ^ atLocation(o2, L1)
0 !wupSimilarity(o1, o2) ^ atLocation(o1, L1) ^ !atLocation(o2, L1)
0 !wupSimilarity(o1, o2) ^ !atLocation(o1, L1) ^ atLocation(o2, L1)
0 !wupSimilarity(o1, o2) ^ !atLocation(o1, L1) ^ !atLocation(o2, L1)

[...]
```

5.2.6.4 Further Modeling Techniques

In contrast to the other classifiers that we discussed, MLNs give us also the expressive power to explicitly model the locations and therefore to define properties and relations of/between the locations themselves.

Reasonable extensions of our models would be to include the types of the locations (e.g. *fridge, drawer, cupboard*) as additional features. We can furthermore include spatial relations between the locations using predicates like *nextTo(l1, l2)* or *nextTo(l1, Oven)*. This would allow us to exploit information about object placement that is given implicitly by the spatial kitchen layout. The power of Markov Logic Networks, as the most general model that we consider here, thereby allow us to incorporate additional knowledge that goes beyond the scope of this work and might help to further improve on our results. We leave this as a topic for future

research as we did not perform an experimental evaluation of MLNs due to the practical challenges described in the next section.

5.2.6.5 Challenges

Although Markov Logic Networks are very powerful and a sound theoretical model, their practical application to non-trivial problems yields many challenges, especially with respect to computational complexity.

- *Learning:*
  Our main challenge with Markov Logic Networks is the computational effort necessary for parameter learning. We performed preliminary tests for the MLNs of different complexities with our soft evidence learning algorithms PLL-ISE, SLL-ISE and DSLL-WW (see Section 4.4.5). As expected, PLL-ISE fails in all but the smallest examples to learn a model that gives reasonable results for our classification task as the pseudo-log-likelihood computation makes additional independence assumptions to simplify the problem. SLL-ISE and DSLL-WW often give more reasonable results, but their runtime is too long to perform a thorough evaluation. Learning MLNs for our simplest model takes several hours to days on a modern desktop machine (Intel Core i7 processor), learning for the more complex models often does not converge in any reasonable timeframe. Although we are able to make a tradeoff between precision and learning time by changing the number of samples drawn in our sampling based methods, we do not yet have knowledge to make a prediction of a reasonable number of samples for a given problem.

  We scale down the weight gradient if it exceeds a fixed threshold to keep the weight values low in early optimization stages (start values are 0), because large weights (approx. > 1) lead to very long MCSAT sampling times. As topic for future research, further speedups might be achieved by using an adaptive number of samples for the sample-based approaches SLL-ISE and DSLL-WW. For the normalization constant $\tilde{Z}$, it could be based on the growth rate of $\tilde{Z}$ during the sampling process, for the sampling of S_e in DSLL-WW on the error between the soft evidence and the relative frequencies in the sampled Markov chain. Furthermore we could improve performance through the parallelization of the MCSAT algorithm, that is used during sampling, by running one Markov chain per thread, as all of them can be sampled independently from each other. This would allow an almost linear scaling with the number of threads, which could give huge speedups if used on large clusters or GPUs.

Although these methods might lead to significant speedups, it remains questionable if MLNs will be applicable to real-world problems like our kitchen classification without new learning methods that reduce the computational complexity while still giving reasonable results.

- *Inference:*
  After learning weights for one of our more complex kitchen models, we experienced MCSAT getting stuck during inference for the *atLocation* predicate (classification task). This happens because of extremely slow convergence of the sampled distribution toward the real probabilities. For any reasonable number of sampling steps, MCSAT will just return a distribution that assigns a probability of 1 to a random location (whose predicate is set to *true* in the state it gets stuck in). This can happen because we have several formulas with positive weights that are conjunctions containing the *atLocation* predicate for a specific location (e.g. in the MLN presented in Section 5.2.6.2). Assume that during the execution of MCSAT, our Markov chain is in a state where some of these conjunctions have the truth value *true*. To switch the state to another location we would have to remove all of them in one sampling step from the set of constraints M (see Algorithm 1). The probability of removing a single formula with weight w from M is e^{-w}. The probability of removing several formulas with positive weights from the set of constraints is therefore very low.

  If we just want to solve our classification task by querying for the *atLocation* predicate for a single object, given the truth values for all other atoms as evidence, we can simply use exact inference. This is feasible because the number of possible worlds in this case is the number of locations $|\mathcal{L}|$ as the truth values for all ground atoms, except the ones relevant for the query, are given by the evidence. To handle soft evidence during exact inference, we use soft counts, as defined in Section 4.4.5.

5.3 Organizational Principles: Feature Importance Measure

In addition to the classification task, we analyze the degree to which features are capable of defining organizational principles at each particular location, fostering an intuitive understanding of the principles (implicitly) represented in a classifier. To this end, the conditional distributions of the attributes given a location as represented in a naive Bayesian classifier trained on the dataset D can be used. Within

a given location L, there is certainly structure with respect to a particular feature F if the conditional distribution of the feature given L exhibits little entropy. Thus, the degree to which F defines an organizational principle at L can be computed as the inverse normalized Shannon entropy,

$$I_F^D(L) := 1 - \frac{\sum_{f \in \mathrm{dom}(F)} P_D(F = f \mid L) \log(P_D(F = f \mid L))}{\log(|\mathrm{dom}(F)|)} \tag{5.4}$$

where $\mathrm{dom}(F)$ is the domain of F, the numerator is the entropy of the distribution over $\mathrm{dom}(F)$ and the denominator is the maximum possible entropy (uniform distribution over F's domain). We thus obtain an importance value in the interval $[0; 1]$, with low values representing low importance (high relative entropy) and high values (low relative entropy) high importance.

In order to analyze the discriminative power of the various aggregated features, we compute the Hellinger distance [29] $H_F^D \in [0; 1]$ between the distributions of a feature F given the locations L_1 and L_2 as follows

$$H_F^D(L_1, L_2) = \sqrt{1 - \sum_{f \in \mathrm{dom}(F)} \sqrt{P_D(F = f \mid L_1) P_D(F = f \mid L_2)}} \tag{5.5}$$

As a measure of distribution dissimilarity, the Hellinger distance is an adequate indicator for feature relevance [30]. We average Hellinger distances across all pairs of locations (from the dataset's set of locations $\mathcal{L}_D$) and define

$$\overline{H}^D(F) := \binom{|\mathcal{L}_D|}{2}^{-1} \sum_{L_i \in \mathcal{L}_D} \sum_{L_j \in \mathcal{L}_D, i<j} H_F^D(L_i, L_j) \tag{5.6}$$

6 Evaluation

We tested our classifiers using two different sets of features on both of our datasets. We first describe our experimental setup and then present and discuss the results.

6.1 Experimental Setup

We performed experiments for the datasets presented in Section 3 using the classifiers defined in Section 5.2. As the learning of reasonable complex Markov Logic Networks turned out to be computationally too expensive in our kitchen scenario (see the challenges described Section 5.2.6.5), we excluded the MLN models from our evaluation.

For each classifier and dataset, we performed two experiments, one using a feature vector containing just the *maxWup* and *avgWup* features and one using a feature vector containing all the features in an effort to determine the power of the WUP similarity when applied to various types of classifiers. In each experiment, we performed leave-one-out cross-validation, i.e. for each class of objects, we removed all of the objects belonging to the class from the kitchen for training and used our classifiers to infer the location at which the object should be stored. A classification result is considered correct if the predicted location is the one at which the object was originally located. If a location contained only objects of a single class, we skipped this class during leave-one-out crossvalidation as our classifiers would have no training samples for that particular location.

6.2 Results and Discussion

We list the mean and standard deviation of the percentage of correctly classified objects (accuracy) for our dataset of mockup kitchens and the accuracy values and their mean for our real kitchen datasets in Table 6.1. The accuracy values for each of the ten kitchens in our mockup kitchen dataset can be found in Table 6.2.

Table 6.1: Results: Accuracy for all classifiers on the ten mockup kitchen datasets

	avgWup and maxWup		**all features**	
	mean	**std**	**mean**	**std**
max. avgWup	77.45%	20.85%	—	—
max. maxWup	87.52%	17.91%	—	—
DecisionTrees	86.61%	12.46%	88.12%	14.16%
Boosted DecisionTrees	87.68%	13.95%	**89.50%**	9.92%
SVM	77.46%	24.11%	89.49%	17.68%
NB Discrete	78.37%	15.64%	85.69%	15.56%
NB Continuous	69.97%	28.63%	82.61%	17.75%
NB Soft	42.16%	39.38%	82.32%	18.73%

Table 6.2: Results: Mean accuracy and standard deviation for all classifiers on both real-world datasets

	avgWup and maxWup			**all features**		
	D_{r1}	D_{r2}	**mean**	D_{r1}	D_{r2}	**mean**
max. avgWup	48.19%	70.24%	59.22%	—	—	—
max. maxWup	72.29%	71.43%	71.86%	—	—	—
DecisionTrees	84.94%	73.81%	**79.37%**	79.52%	69.05%	74.28%
Boosted DecisionTrees	84.94%	71.43%	78.18%	80.72%	69.05%	74.89%
SVM	57.23%	69.05%	63.14%	73.49%	76.19%	74.84%
NB Discrete	50.00%	50.00%	50.09%	57.83%	64.29%	61.06%
NB Continuous	41.57%	58.33%	49.95%	60.24%	63.10%	61.67%
NB Soft	13.86%	50.00%	31.93%	65.66%	59.52%	62.59%

Table 6.3: Results: Accuracy for all classifiers on each of the ten kitchens in our mockup kitchen dataset

		D_{m1}	D_{m2}	D_{m3}	D_{m4}	D_{m5}
max. avgWup	WUP	81.82%	71.88%	75.76%	72.31%	77.27%
max. maxWup	WUP	90.91%	92.19%	90.91%	76.92%	80.30%
DecisionTrees	WUP	89.39%	89.06%	84.85%	83.08%	80.30%
DecisionTrees	all	93.94%	87.50%	84.85%	81.54%	84.85%
Boosted DecisionTrees	WUP	92.42%	90.63%	84.85%	84.62%	80.30%
Boosted DecisionTrees	all	92.42%	90.63%	86.36%	87.69%	84.85%
SVM	WUP	83.33%	79.69%	75.76%	63.08%	65.15%
SVM	all	95.45%	85.94%	84.85%	89.23%	78.79%
NB Discrete	WUP	84.85%	75.00%	84.85%	75.38%	69.70%
NB Discrete	all	93.94%	89.06%	87.88%	78.46%	80.30%
NB Continous	WUP	68.18%	60.94%	59.09%	58.46%	63.64%
NB Continous	all	92.42%	70.31%	81.82%	84.62%	81.82%
NB Soft	WUP	30.30%	37.50%	19.70%	49.23%	34.85%
NB Soft	all	89.39%	75.00%	77.27%	83.08%	74.24%

		D_{m6}	D_{m7}	D_{m8}	D_{m9}	D_{m10}
max. avgWup	WUP	63.64%	84.85%	77.27%	84.85%	84.85%
max. maxWup	WUP	95.45%	84.85%	87.88%	83.33%	92.42%
DecisionTrees	WUP	93.94%	83.33%	84.85%	86.36%	90.91%
DecisionTrees	all	95.45%	86.36%	87.88%	84.85%	93.94%
Boosted DecisionTrees	WUP	95.45%	83.33%	86.36%	87.88%	90.91%
Boosted DecisionTrees	all	93.94%	90.91%	87.88%	86.36%	93.94%
SVM	WUP	78.79%	83.33%	74.24%	86.36%	84.85%
SVM	all	98.48%	90.91%	89.39%	86.36%	95.45%
NB Discrete	WUP	80.30%	77.27%	72.73%	80.30%	83.33%
NB Discrete	all	81.82%	83.33%	81.82%	89.39%	90.91%
NB Continous	WUP	81.82%	77.27%	68.18%	81.82%	80.30%
NB Continous	all	80.30%	86.36%	81.82%	78.79%	87.88%
NB Soft	WUP	37.88%	42.42%	46.97%	62.12%	60.61%
NB Soft	all	93.94%	86.36%	81.82%	80.30%	81.82%

From these results, we conclude that the features based on WUP similarity are, indeed, highly discriminative. Using the maximum *maxWup* similarity alone yields an accuracy of 88% and 72% for the mock-up and real kitchens respectively. This coincides with the human intuition that placing an object at the location where the most similar object is located would be a reasonable strategy.

In our dataset containing the ten kitchen mockups, adding additional features and using more sophisticated classifiers like SVMs, (boosted) decision trees and naive Bayes yields only a small improvement, if any, with boosted decision trees using all features yielding the best results (90% accuracy).

In the more complex scenario of the real kitchen dataset, however, (boosted) decision trees and SVMs are able to improve upon the simple *maxWup* classifier by up to 8%. It is interesting to note that we get the best results with decision trees using only the WUP similarities. The semantic similarity measure is sufficiently powerful to obtain a correct classification rate of 79% in a real kitchen. In many ways, the real-world environment is a more complex scenario, because object placement may be influenced by additional spatial restrictions and convenience considerations (e.g. highly similar objects are not always placed at a single location because no container is large enough to hold them all). Such constraints were not considered in our mockup experiments.

It is important to realize that a classification rate close to 100% may not be possible in practice. First, a limited subset of objects in real-world environments may indeed have been placed arbitrarily (for reasons such as lack of time or laziness), inducing noise in the data. The underlying principles would therefore be neither possible nor desirable for us to model. Second, several objects of one class could reasonably be placed at more than one location, yet our evaluation considers only one of them as correct. Given probabilistic models like our naive Bayesian classifiers, we could have considered locations that received a probability higher than a certain threshold as correct, but we did not follow this approach because the results would hardly have been comparable to the other classifiers.

We now analyze the degree to which there is discernable organizational structure at the various locations in a kitchen. In Figure 6.1, we present a plot showing the importance measure defined in Section 5.3 for one of the kitchens from our mockup dataset. A photograph of that kitchen layout is shown in Figure 3.1. We reduced the importance values for our twelve *avgWup*, twelve *maxWup*, four *purpose* and five *purposeMeal* features to four average values, one for each set of features. The plot shows that all features except size are very prevalent at location 3. In our dataset, this seems reasonable, because the location contains cooking pots and pans, which are similar with respect to most aspects with the exception of size. At location 2, feature importance values are low, because it represents the fridge, which

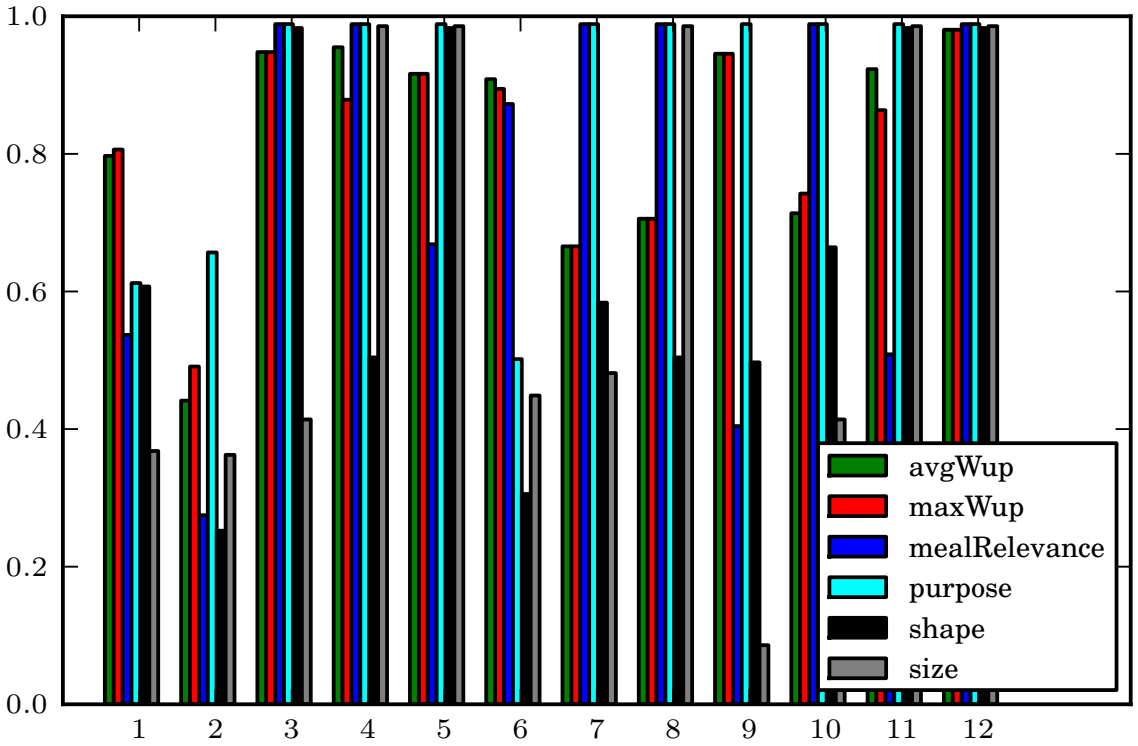


Figure 6.1: Average feature importance ($I_F^D(L)$) for the 12 different locations (x-axis) in the mockup kitchen dataset. High values indicate structure at a particular location with respect to a particular group of features.

contains the greatest variety of different products. Looking at the prevalent feature values at locations with high importance gives us some human-readable indication of the organization principles, e.g. "FoodVessel, PhysicalDevice, PrincipalMeal, shape: other" for location 3 or "FoodOrDrink, Breakfast" for location 4, which contains breakfast cereals.

In order to analyze the discriminative power of the various features, we present in Figure 6.2 and Figure 6.3 the Hellinger distances of feature distributions averaged across all pairs of locations in our mockup kitchen dataset and our real-world kitchen dataset respectively. We observe that, as expected, the WUP similarities have the highest average Hellinger distances for both datasets, which means that among the features we considered, they are best-suited for a discrimination between places.

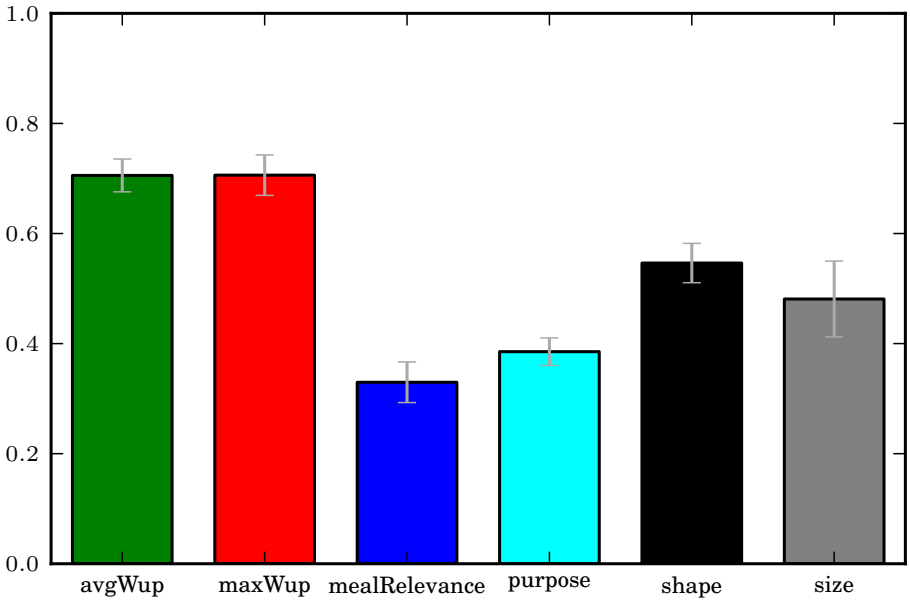


Figure 6.2: Mean and standard deviation of the Hellinger distance-based importance measure $\overline{H}^D$ (defined for feature groups by taking the average) for the ten kitchens in our mockup kitchen dataset

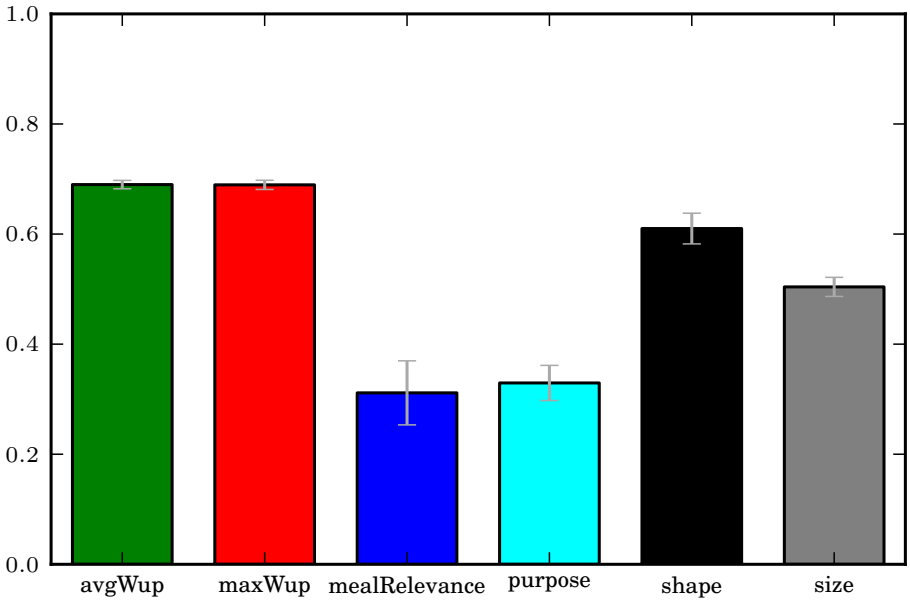


Figure 6.3: Mean and standard deviation of the Hellinger distance-based importance measure $\overline{H}^D$ (defined for feature groups by taking the average) for the two kitchens in our real kitchens dataset

7 System Integration

We provide an open source implementation of our algorithm to solve the object allocation task as an extension to the KnowRob [1] knowledge processing system. We give an overview over the system in Figure 7.1. A robot performing a high-level reasoning task can use an ask-and-tell interface, based on Prolog predicates, to query information from the knowledge base. In our case here, this is the query for a location where to place or search for a particular object. The necessary background information for our algorithm is given by the KnowRob kitchen ontology, including the *germandeli* data, and information about the locations of objects, gathered by the robot's perception system.

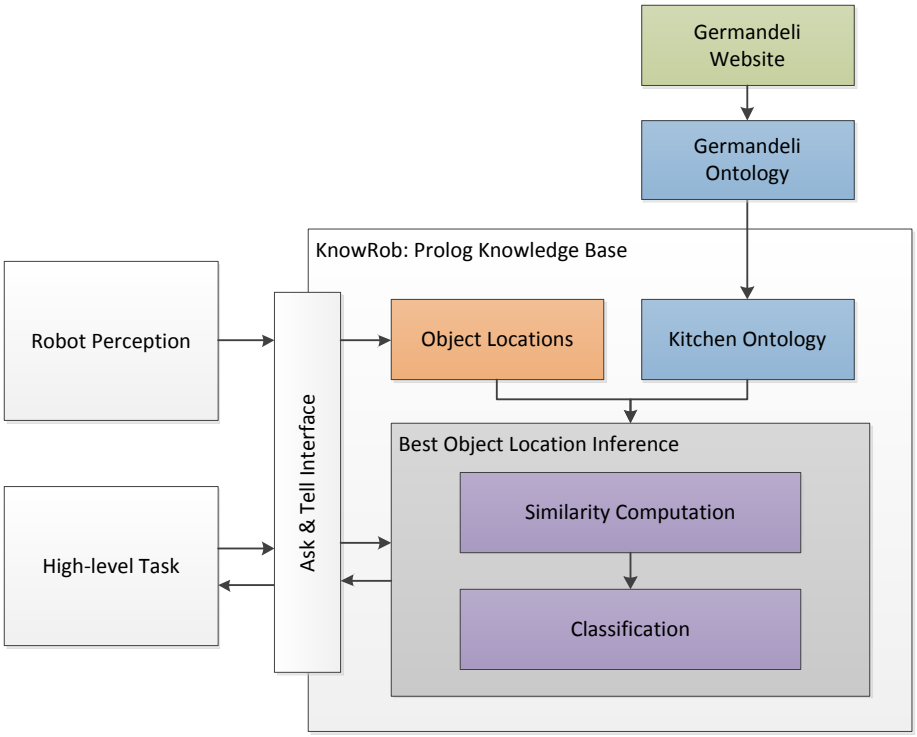


Figure 7.1: System overview: Integration of our algorithm in the KnowRob knowledge processing system

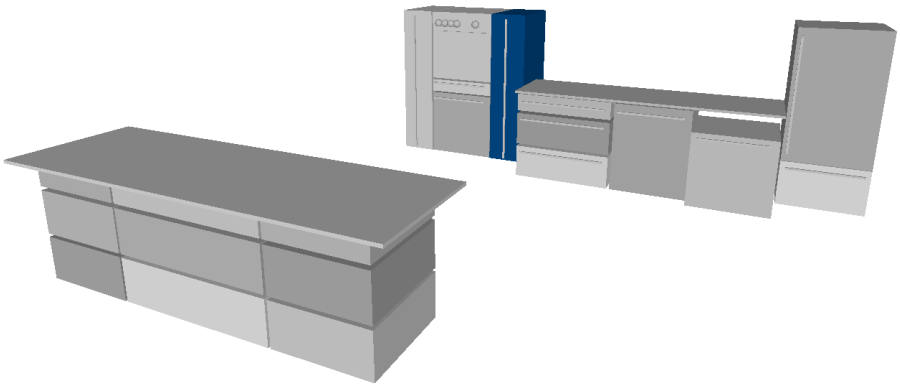


Figure 7.2: Lab kitchen (KnowRob 3d visualization), the inferred best storage location for coffee filters is highlighted in blue

We provide several Prolog predicates for WUP similarity computation and, based on that, predicates for the inference of the best location for a given object[1]. In particular, we have two predicates, *best_location_maxMaxWup(Object, Location)* and *best_location_dtree(Object, Location)*, that compute the best location where to place or search for an object applying our maximum *maxWup* classifier and our decision tree classifier with the WUP similarity based feature set, respectively. In our evaluation (see Section 6) we showed that these classifiers exhibited good performance in all scenarios.

Within the KnowRob system, we can visualize the results by highlighting the chosen location in a 3d representation of our lab kitchen. The following Prolog query shows where to best place a package of coffee filters. The corresponding object instance is named *orgprinciples_demo:CoffeeFilter1*, its class, defined within the *germandeli* ontology, is *germandeli:Kaffeefilter_Nr_4_80pc*.

```
?- highlight_best_location_dtree(
     orgprinciples_demo:'CoffeeFilter1', $Canvas).
```

The visualization is presented in Figure 7.2. The console output lists all the objects known to be located at the inferred location as well as their classes and the WUP similarities to *orgprinciples_demo:CoffeeFilter1*:

```
Best location: knowrob:Drawer7
Objects at location knowrob:Drawer7:
WUP similarity: object (class)
```

[1]Our implementation is split over two ROS (Robot Operating System, http://www.ros.org/) packages. We provide Prolog predicates for WUP similarity computation in the *ias_prolog_addons* package and Prolog predicates to infer and visualize object locations in the *comp_orgprinciples* package

```
0.70588: orgprinciples_demo:ChocolateDrink1
  (germandeli:Milka_Schoko_Drink_refill_500g)
0.87500: orgprinciples_demo:CoffeGround1
  (germandeli:Dallmayr_Classic_Ground_Coffee_250g)
0.87500: orgprinciples_demo:CoffeeGround2
  (germandeli:Dallmayr_Extra_Spezial_Ground_500g)
0.75000: orgprinciples_demo:EspressoBeans1
  (germandeli:illy_Espresso_Whole_Beans_88_oz)
0.70588: orgprinciples_demo:Sugar1
  (germandeli:Nordzucker_Brauner_Teezucker_500g)
0.66667: orgprinciples_demo:Tea1
  (germandeli:Teekanne_Rotbusch_Tee_20_Bags)
0.66667: orgprinciples_demo:Tea2
  (germandeli:Teekanne_Rotbusch_Tee_Vanille_20_Bags)
```

All of these objects exhibit a high similarity to coffee filters because they are all usually used to prepare (hot) drinks, including coffee.

We now want to position our algorithm in the context of a higher-level task with the help of our aforementioned examplary scenario. Therefore assume someone (or some robot) just came home from shopping in a supermarket, places a basket full of newly bought items on the kitchen table and tells our assistive kitchen robot to "put the things away", i.e. to place them at reasonable locations within the kitchen. This may result in the following steps for the robot:

1. *Empty shopping basket on the table to be able to separate the objects*

2. *Perceive and segment objects*

3. *Match objects with object classes in knowledge base*
   In case an object is unknown, this can require to gather additional information, for example from the internet by parsing online shop websites (e.g. *germandeli.com*)

4. *Use general knowledge about kitchens to handle certain types of objects*
   This knowledge can be represented by hardcoded rules or learned by approaches similar to the ones presented in our related work (see Section 1.4). Such hard rules could for example require to place frozen objects into the freezer. General knowledge about room types could be used to infer which of the objects belong into the kitchen at all.

5. *Use our new algorithm to infer where to best place the (remaining) kitchen objects*

The result will be specific to this particular kitchen and the organizational principles implicitly imposed by its owner through previous object placements. Our algorithm uses similarities to other objects that are already located in the kitchen to infer the best location. The robot therefore needs to aquire information about the other locations of the objects in the kitchen once in a while (or alternatively observe and remember all changes). This information does not need to be 100% correct or up to date though, because organizational principles are usually not changed very often.

6. *For each object:*

   a) *Pick up the object*

   b) *Move to inferred location, open container if necessary*

   c) *Search for free space inside location, if there is not enough free space, infer a new location (→ Step 5)*

   d) *Place object inside location, close container if necessary*

8 Conclusion and Future Work

In this work, we have addressed the important task of identifying structure in human living environments – a task that will become increasingly relevant as robots begin to assume their role as household assistants under real-world conditions. We considered the task of allocating objects to likely storage locations, which can reasonably be viewed as a classification task, and analyzed the suitability of various classifiers for this task. Our thesis that organizational principles can be viewed as manifestations of clusterings that are governed by similarity was confirmed in our experiments, and the semantic similarity measure that we proposed based on *WUP similarity* proved to be highly informative. Average classification rates of at least 79% could be reached even in real-world scenarios, and standard classifiers such as support vector machines and decision trees proved to be adequate.

Naive Bayesian classifiers performed less reliably, but we could exploit their probabilistic semantics to gain additional information about the structure of the problem and the relevance of our various features. Markov Logic Networks, although in theory our most powerful model and capable of handling additional types of features and queries, turned out to be computationally too extensive to be applicable on our datasets. We developed the foundations of new soft evidence weight learning methods, but to apply these methods to handle non-trivial scenarios, further optimizations to reduce their computational complexity will be necessary.

We provide an open source implementation of the best performing variant of our algorithm, integrated into the KnowRob knowledge processing system. We thereby provide Prolog predicates to infer the best location where to place or search for objects, which can easily be incorporated in higher-level reasoning tasks.

In future work, we intend to include more features that will consider, for example, spatial relations between the containers in an environment and more fine-grained representation of locations, such that we can model the concrete spatial configuration within specific locations. Furthermore, we plan to address the problem of transferring our approach to the learning of organizational principles in other everyday environments. We are currently working on the integration of our models in a robotic system that is to solve complex everyday tasks within a kitchen environment.

Bibliography

[1] M. Tenorth and M. Beetz, "KnowRob — Knowledge Processing for Autonomous Personal Robots," in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, 2009.

[2] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of Cyc," *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pp. 44–49, 2006.

[3] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled Robots – Robots that use the Web as an Information Resource," *Robotics & Automation Magazine*, vol. 18, no. 2, 2011, accepted for publication.

[4] D. Jain, P. Maier, and G. Wylezich, "Markov Logic as a Modelling Language for Weighted Constraint Satisfaction Problems," in *Eighth International Workshop on Constraint Modelling and Reformulation. In conjunction with CP2009.*, 2009.

[5] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J. Fernández-Madrigal, and J. González, "Multi-hierarchical semantic maps for mobile robotics," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Edmonton, CA, 2005, pp. 3492–3497.

[6] K. Sjöö, H. Zender, P. Jensfelt, G.-J. M. Kruijff, A. Pronobis, N. Hawes, and M. Brenner, "The explorer system," in *Cognitive Systems*, H. I. Christensen, G.-J. M. Kruijff, and J. L. Wyatt, Eds. Springer, 2010.

[7] A. Bouguerra, L. Karlsson, and A. Saffiotti, "Handling uncertainty in semantic-knowledge based execution monitoring," in *IROS*. IEEE, 2007, pp. 437–443.

[8] H. Zender, O. Martínez Mozos, P. Jensfelt, G. Kruijff, and W. Burgard, "Conceptual spatial representations for indoor mobile robots," *Robotics and Autonomous Systems*, 2008.

[9] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart, "Cognitive maps for mobile robots - an object based approach," *Robotics and Autonomous Systems*, vol. 55, no. 5, pp. 359–371, 2007.

[10] S. Vasudevan and R. Siegwart, "Bayesian space conceptualization and place classification for semantic maps in mobile robotics," *Robotics and Autonomous Systems*, vol. 56, no. 6, pp. 522–537, 2008.

[11] J. Sinapov and A. Stoytchev, "The odd one out task: Toward an intelligence test for robots," in *IEEE 9th International Conference on Development and Learning (ICDL)*, 2010, pp. 126 –131.

[12] D. D. Fu, K. J. Hammond, and M. J. Swain, "Action and perception in man-made environments," in *Proc. of the 14th IJCAI*, Montreal, Canada, 1995, pp. 464–469.

[13] N. J. Smith, *Vagueness and Degrees of Truth*. Oxford University Press, 2008.

[14] M. Broecheler, L. Mihalkova, and L. Getoor, "Probabilistic similarity logic," in *Conference on Uncertainty in Artificial Intelligence*, 2010.

[15] D. Jain and M. Beetz, "Soft Evidential Update via Markov Chain Monte Carlo Inference," in *33rd Annual German Conference on Artificial Intelligence (KI 2010)*, 2010.

[16] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI-01 workshop on "Empirical Methods in AI"*.

[17] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[18] R. R. Bouckaert, "Naive bayes classifiers that perform well with continuous variables," in *Australian Conference on Artificial Intelligence*, ser. Lecture Notes in Computer Science, G. I. Webb and X. Yu, Eds., vol. 3339. Springer, 2004, pp. 1089–1094.

[19] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[20] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla, "Markov logic," in *Probabilistic Inductive Logic Programming*, ser. Lecture Notes in Com-

puter Science, L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, Eds., vol. 4911. Springer, 2008, pp. 92–117.

[21] D. Jain, "Applying markov logic to the acquisition of and reasoning about action models," *Diploma thesis in Computer Science at the Technische Universität München*, 2007.

[22] V. Gogate, W. Webb, and P. Domingos, "Learning efficient markov networks," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., 2010, pp. 748–756.

[23] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[24] W. Wei and B. Selman, "A new approach to model counting," in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 324–339.

[25] D. Lowd and P. Domingos, "Efficient weight learning for markov logic networks," in *PKDD*, ser. Lecture Notes in Computer Science, J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic, and A. Skowron, Eds., vol. 4702. Springer, 2007, pp. 200–211.

[26] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, H. Poon, D. Lowd, J. Wang, and A. Nath, "The Alchemy System for Statistical Relational AI," http://alchemy.cs.washington.edu/, 2010.

[27] Z. Wu and M. S. Palmer, "Verb semantics and lexical selection," in *ACL*, 1994, pp. 133–138.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[29] L. L. Cam and G. L. Yang, *Asymptotics in Statistics – Some Basic Concepts.* Berlin: Springer, 1990.

[30] W. Li and X. Jia, "Feature selection algorithm based on hellinger distance," *Journal of Computer Applications*, vol. 30, no. 6, pp. 1530–1532, 2010.