Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor thesis

Relational Machine Learning in Classification of Cadastral Map Objects

Tomáš Trnka

Supervisor: Radomír Černoch, MSc.

Study Programme: Open Informatics

Field of Study: Computer and Information Science

May 23, 2013

Declaration

I declare that I created this thesis on my own and I quoted every used information source. This was done according to the methodical directive about keeping ethical principles during preparation of final projects at university.

V Praze dne 23. 5. 2013 .....

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodřování etických principů při přípravě vysokoškolských závěřečných prací

V Praze dne 23. 5. 2013 .....

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš T r n k a

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Relational Machine Learning in Classification of Cadastral Map Objects

Guidelines:

1. Review the relevant literature.
2. Choose an appropriate formalism of relational machine learning with uncertainty.
3. Create an application for downloading the maps provided by the Czech Office for Surveying, Mapping and Cadastre and their segmentation. Formulate the classification task in the above chosen formalism.
4. Evaluate the classifier.

Bibliography/Sources:

[1] De Raedt, L. : Logical and Relational Learning. Springer, 2008.
[2] Richardson, M.; Domingos, P.: Markov Logic Networks. Journal Machine Learning 62, pages 107-136, Kluwer Academic Publishers Hingham, MA, USA, 2006.
[3] Navara, M.: Základy fuzzy množin. Nakladatelství ČVUT Praha, 2007.
[4] Straccia, U.: Reasoning within Fuzzy Description Logics. JAIR, vol. 14, pages 137- 166, 2001.
[5] Bobillo, F. and Straccia, U.: fuzzyDL: An Expressive Fuzzy Description Logic Reasoner. IEEE World Congress on Computational Intelligence, pp.923-930, 1-6 June 2008.
[6] Mayer, H.: Automatic Object Extraction from Aerial Imagery - A Survey Focusing on Buildings. Computer Vision and Image Understanding, vol. 74 (2), pp. 138-149, 1999.
[7] Baltsavias, E. P.: Object extraction and revision by image analysis using existing geodata and knowledge: current status and steps towards operational systems. ISPRS Journal of Photogrammetry and Remote Sensing, vol. 58, iss. 3–4, pp.129-151,January 2004.

Bachelor Project Supervisor: Radomír Černoch, MSc.

Valid until: the end of the winter semester of academic year 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Tomáš T r n k a

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Relační strojové učení v klasifikaci objektů katastrálních map

Pokyny pro vypracování:

1. Proveďte rešerši relevantní literatury.
2. Zvolte vhodný systém relačního učení s neurčitostí.
3. Vytvořte aplikaci pro stažení map Českého úřadu zeměměřického a katastrálního a jejich segmentaci. Ve výše zvoleném systému zformulujte klasifikační úlohu.
4. Vyhodnoťte úspěšnost navrženého klasifikátoru.

Seznam odborné literatury:

[1] De Raedt, L. : Logical and Relational Learning. Springer, 2008.
[2] Richardson, M.; Domingos, P.: Markov Logic Networks. Journal Machine Learning 62, pages 107-136, Kluwer Academic Publishers Hingham, MA, USA, 2006.
[3] Navara, M.: Základy fuzzy množin. Nakladatelství ČVUT Praha, 2007.
[4] Straccia, U.: Reasoning within Fuzzy Description Logics. JAIR, vol. 14, pages 137- 166, 2001.
[5] Bobillo, F. and Straccia, U.: fuzzyDL: An Expressive Fuzzy Description Logic Reasoner. IEEE World Congress on Computational Intelligence, pp.923-930, 1-6 June 2008.
[6] Mayer, H.: Automatic Object Extraction from Aerial Imagery - A Survey Focusing on Buildings. Computer Vision and Image Understanding, vol. 74 (2), pp. 138-149, 1999.
[7] Baltsavias, E. P.: Object extraction and revision by image analysis using existing geodata and knowledge: current status and steps towards operational systems. ISPRS Journal of Photogrammetry and Remote Sensing, vol. 58, iss. 3–4, pp.129-151,January 2004.

Vedoucí bakalářské práce: Radomír Černoch, MSc.

Platnost zadání: do konce zimního semestru 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

Acknowledgements

I would like to thank to my supervisor Mr. Radomír Černoch for his support and encouragement while writing this thesis. I also would like to thank to my family for their generous support allowing me to study.

Abstract

This thesis describes a classifier of objects on a real world cadastre map within the relational machine learning framework.
The first part of the work aims to introduce and compare relational probabilistic reasoners that allows us to create a model over the data that would be interpretable and corresponding to the human description of the objects on the map.
For the implementation part it is necessary to execute some basics image processing to obtain some probabilistic facts and allow us to create a model of the data.
There are several state of the art classifier allowing to classify object recognized by computer vision. We tried to create one that takes into account reasoning the relations between the object than rather handling them as independent.

Index Terms: relational learning, machine learning, probabilistic reasoners, classification, cadastral maps

Abstrakt

V této práci jsme se pokusili vytvořit model pro rozhodování ve skutečných katastrálních mapách na základě relačního strojového učení.
V první části prace je srovnání existujících pravděpodobnostních reasonerů, které by nám umožnili vytvořit nad daty takový model, který by byl srozumitelný i pro lidi a popisoval objekty na mapách, tak jak by to dělal člověk.
V implementační části bylo nejprve nutné získat z obrázků katastrálních map data, která by bylo možné takovým způsobem vyjádřit. Pro vyhodnocení těchto dat existuje mnoho klasifikátorů, které bychom mohli použít. Naší snahou však bylo vytvořit takový, který by bral v potaz vztahy mezi objekty.

Klíčová slova: relační učení, strojové učení, pravděpodobnostní resonery, klasifikace, katastrální mapy

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this work I would like to describe some approaches to make a connection between ILP and statistics. There are several existing solutions implemented so far. In the first part I have compared the approaches and discussed the most appropriate one for my task.

Namely, the implementations are MLN [16] and ProbLog [15], and other technology based on Fuzzy logic. In this thesis a new model should be built that could be used for classification of a map.

The aim of this work was to compare the state of the art probabilistic reasoners, and choose one of them that fulfil our needs. Then we intended to process some real world data. Then we tested the quality, the speed of the chosen reasoner. The next step we were planning to execute was exploring the possibility of creating relational features and how much they influence the results. One more question was how much we can improve reasoning about the objects using this knowledge in comparison to the other classification applications that use an attribute-value learner.

The goal was to build a functional application capable to classify, but not in the first place. This work is rather an attempt to analyse the problem, endeavour to build model and reason about it. The result that we care about most is whether it is even possible to use state of art reasoner to solve this task, examine how complicated and complex the model could be to obtain the results in reasonable time with sufficient accuracy.

Chapter 2

Map recognition

In this chapter I will provide a brief description of the image processing programs and the outputs that are used for creating the logical model of the data. This process is described from acquiring to processing the image. The next section covers the description of datasets.

2.1 Used tools

There was a strong preference to use free software to allow the free availability of the work's results. The following list provides information of the used tools and libraries in this work.

- Python can be according the Wikipedia article [3] described as 'a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.'

- Numpy [1] 'is a package for scientific computing with Python. It contains among other things a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ code and linear algebra, Fourier transform, and random number capabilities.'

- SciPy [4] 'is an open-source software for mathematics, science, and engineering. It is also the name of a very popular conference on scientific programming with Python. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.'

- Mahotas [8] 'is a set of well documented functions for image processing and computer vision in Python. It is based on numpy arrays as its datatype. It has its heavy routines

implemented in clean C++ in a way that is both very clean, type independent (using templates), and fast.'

- Python Imaging Library (PIL) [2] 'adds image processing capabilities to Python. This library supports many file formats, and provides powerful image processing and graphics capabilities.'
- Web Map Service (WMS) [5] 'is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database.'

2.2 Image acquiring and processing

The images for processing are downloaded from the web pages of Czech Office for Surveying, Mapping and Cadastre. They provide WMS api for downloading. Please take note that the image processing part is not the core part of this paper. There is a lot of space for improvements. The main reason for the explanation of image processing is the repeatability of our experiment and making the description complete.

2.2.1 Building WMS query

The Czech Office for Surveying, Mapping and Cadastre is accessible via the web page but for purpose of downloading bigger data there is a WMS api. The WMS request is an HTTP request and the server responds with the desired image. The request must specify map layers, coordinate system, type and size of the image and the requested area. For obtaining the image all these choices must be specified in a GET request and then send to the server. For downloading the image from WMS server we used a query that looked like this:

Listing 2.1: WMS query

```
1 http://services.cuzk.cz/wms/wms.asp
2 ?LAYERS=dalsi_p_mapy,hranice_parcel,obrazy_parcel,omp
3 &TRANSPARENT=FLASE&FORMAT=image/jpeg&VERSION=1.3.0
4 &EXCEPTIONS=INIMAGE&SERVICE=WMS&REQUEST=GetMap
5 &CRS=EPSG:32633
6 &WIDTH=%d&HEIGHT=%d&BBOX=%d,%d,%d,%d
```

The interesting lines are 2, 5 and 6. The second line specifies layers to download. The line 5 determines selected coordinate system. In the sixth line parameters for height, width and limits for BBOX has to be filled in. The parameters are derived from the coordinate system and the level of desired zoom. For converting from longitude and latitude data into the WGS84 coordinate system we have rewritten a publicly accessible JavaScript code [9] into Python.

2.2.2 Creating the resulting image

There are several limitations for obtaining the images regarding the size, its maximal size is only 4000x4000 pixels. This limitation is solved with tiling. Firstly the WMS queries for desired tiles are built, than the tiles are downloaded and afterwards stitched together with PIL.

2.2.3 Reading image and segmentation

The mahotas package provides methods for reading several images into data structures suitable for further processing. The image is converted to a multidimensional Numpy array.

The image is processed with multidimensional Gaussian filter for noise filtering. The standard-deviation is as low as possible because the images are black and white only and there is no other shape then simple line in the images. Then the mahotas library function OTSU thresholding computes [14] the optimal threshold. The threshold is used to separate the regions simply by comparing the value of the Numpy array of given Then the SciPy method `label` is used to separate the regions. The pseudocode looks as follows:

Listing 2.2: Segmentation and image labelling

```
int[n][m] im = read_image_from_file(image);
int[n][m] im2 = compute_filetered_image(im);
int T = determine_threshodl_otsu(im2);
int[n][m] labeled = labled_regions_in_picture(im2 > T);
```

2.2.4 Getting independent features

The labelled array is processed in the foreach loop, each object recognized on the map is 'measured'. The Numpy array method can count the number of cells of the array that are filled with the appropriate label. Thanks to the mahotas library, we are able to fill every object to its convex hull and then obtain its size in the same way. The Numpy also provides methods to determine the maximal position for each label. There is a problem to use this function according to its specification, and therefore we created our own function yielding these results. An approximate circumference of the object can be derived from this feature. There is also a possibility to compute the center of mass for each object. The pseudocode for this part for every label looks like:

Listing 2.3: Acquisitions of features

```
bool[n][m] selected_label = select_label_in_image(labeled,label_number);
int volume = count_true_cells_in_array(selected_label);
bool[n][m] selected_label_convexhull = fill_convexhull(selected_label);
int volume2 = count_true_cells_in_array(selected_label_convexhull);
int[2] = compute_center_of_mass(selected_label);
```

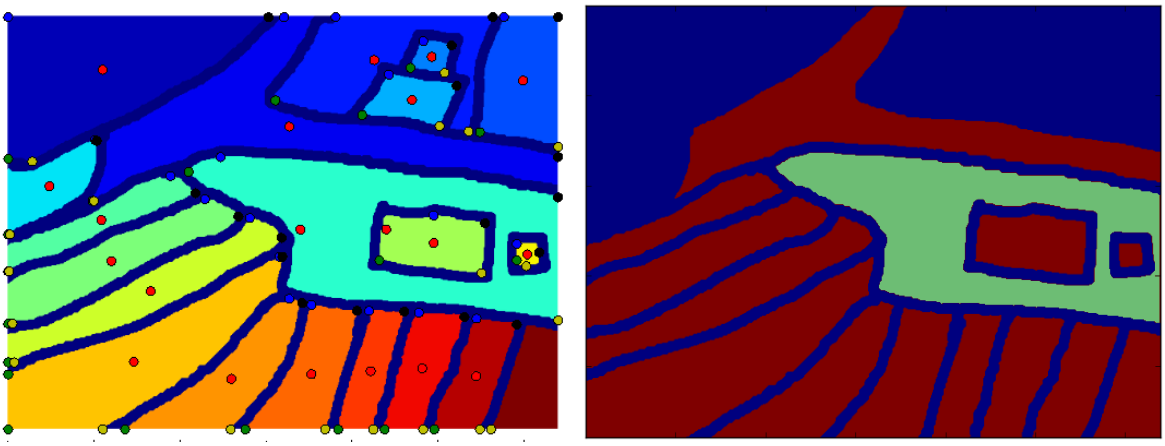
The part finding 'corners' is implemented as:

Listing 2.4: Finding ‘corners’

```
int[2][count_of_labels] top_coordinates;
int[2][count_of_labels] left_coordinates;
int[2][count_of_labels] right_coordinates;
int[2][count_of_labels] bottom_coordinates;
for i<n:
  for j<m:
    if ‘first column in line where the label first appeared’ then:
      top_coordinates[0][labeled[i][j]] = i;
      top_coordinates[1][labeled[i][j]] = j;
    if ‘last column in line where the label appeared at least’ then:
      bottom_coordinates[0][labeled[i][j]] = i;
      bottom_coordinates[1][labeled[i][j]] = j;
    if ‘last row in column where the label appeared at first’ then:
      left_coordinates[0][labeled[i][j]] = i;
      left_coordinates[1][labeled[i][j]] = j;
    if ‘first row in column where the label appeared at least’ then:
      right_coordinates[0][labeled[i][j]] = i;
      right_coordinates[1][labeled[i][j]] = j;
```

Then the result of this processing is shown on figure 2.1a bellow.

Figure 2.1: Processed features



(a) The output of processing

The output of processing, the blue dots representing the top corner, the yellow dots the bottom one. The greens stands for left corners and the blue ones for right corners. The red dots are positions of centres of masses.

(b) Showing selected label and its neighbours

The green area represents the selected label, the red colour stand for each neighbour and the rest of the picture is blue because it is either object not in this relation or a border between two objects.

2.2.5 Getting relational features

The two relational are relations 'next to' and 'is in'. This process of creating this relation is shown in the picture 2.1b. The part of the image is selected with respect to the label. Every point in the selected area is being processed. The algorithm searches for the nearest pixel in the image with a different label. Then the neighbours are identified — see figure 2.1b for results. The relation 'is in' is assumed in case that one object is fully included in a convex hull of the other object. The relation 'next to' helps to prune the search space of object resulting that only the neighbouring objects can contain selected label. Processing these features could be heavily improved - for example rewriting this part of code from Python to C++ can speed up the process up to 10 times.

2.3 Datasets

When creating the dataset we would like to avoid overfitting one special case; therefore, structure of the maps had been separated in 3 categories and there are 2 datasets in each category. The first one in *center of a city*, in our case Prague and Brno. The second category is *microdistrict* here represented by a part of Prague city district 'Jižní Město' and a part of the city Most. The last example is *small villages* - namely 'Boží Dar' and 'Šlapanice'.

The dataset contains six maps; there are two of them of each kind for testing purposes. The map images size ranges from 10000x10000 to 10000x20000 points and contains from 2000 to 3800 recognized objects. The precise number can be found in table 2.1.

The images are included on the attached DVD in form of pictures and also as text files. The text file is a labelled array where each cell of the picture is represented by number and their values are separated from each other with ';'. One line of pixel in image corresponds with one line outputed into text file. For illustration the part of the 'text' image:

Listing 2.5: Labeled text output

```
... 67;67;0;0;0;68;68;68;68;...;68;68\n
```

The data from the processed image are stored in several files. The files `parcels` and `convex_hulls` contain on each line the identifier of the object, i.e. its label and one number that is computed as mentioned above. The example:

Listing 2.6: Text data regarding size

```
p1078 23444\n
p1079 1233\n
```

The maximal and minimal coordinates can be found in files whose name corresponds with the location of the point. The file called with the file name prefix `top`,`bottom`,`left`,`right` respectively. The file contains two number on each row representing x and y coordinate. The last file named `cm` is containing coordinates of the centre of the mass for each object in the same format. Structure of the documents looks like:

Listing 2.7: Text data – coordinates

```
p1078 230 300\n
p1079 123 900\n
```

We have classified each dataset and separated object on the picture into 3 categories. These are ‘street’; ‘house’; ‘parcel’. The classified object are stored during the classification in a set that helps avoid doubling the same data. The data are stored as Prolog facts. Each object has its own label.

Listing 2.8: ProbLog data encoding

```
house(p2382).
parcel(p1137).
street(p1228).
house(p1720).
```

In the data downloaded from Czech Office for Surveying, Mapping and Cadastre there are several object that can be considered as noise. Every building has an identification shown as a dot. There are also other signs for other objects as shown on picture 2.2. These signs

Figure 2.2: Cadastre signs
in order from left to right: garden, orchard, barren land, cultural heritage, building

are in the maps for better human readability, to quickly distinguish from several types of object. In our case these signs are not used for recognizing the object and therefore can be considered redundant. Because they are not the objects we would like to classify, here in the results they have been ignored.

The next table 2.1 includes basic information about each dataset.

name	type	properties				
		objects	ignored o.	width	height	map size
Brno	city center	2189	311	10 000	10 000	715x640m
Praha centrum	city center	2527	577	14 000	10 000	800x620m
Boží dar	countryside	2072	1188	13 600	7 424	900x500m
Šlapanice	countryside	3814	1070	10 000	10 000	650x670m
Most	microdistrict	2473	286	14 000	10 000	1200x630m
Praha Jižní Město	microdistrict	2337	853	14 000	10 000	1000x650m

Table 2.1: Datasets information

I was not able to create bigger image than 20000x20000 due to insufficient amount of testing PC’s RAM(8 GB).

Chapter 3

Probabilistic reasoners

This chapter contains a brief description of all mentioned probabilistic reasoners. In the course of the chapter all the reasoners are compared, showing their similarities and differences. There are also small examples explaining at some basic level how to formulate the task and the interpretation of the results.

Our intention was to build an interpretable model of the data. Our aim was to build a model that would be understandable by a human being; therefore, we are dealing with concepts in the area of relational learning and knowledge representation. This property is not axiomatic for every reasoner. This goal can be fulfilled by any of those reasoners, that is why we included into consideration even the FuzzyDL reasoner.

The next section covers our attempt to create probabilistic data that could be used for reasoning with chosen reasoner.

3.1 ProbLog

This system is a probabilistic extension of the programming language Prolog.

The main assumption is, that all the probabilistic data in our program (KB) are mutually independent. When we ask ProbLog about a probability of a query, then the probability can be interpreted as a probability that the query succeeds in the subprograms of the program. A ProbLog program $T = \{p_1 : c_1, \ldots, p_n : c_n\}$ defines probability distribution over logic programs $L \in L_T = c_1, \ldots, c_n$ in the following way:

$$P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i) \tag{3.1}$$

To compute the probability we have to transform the data. We have to create boolean DNF formula, in which each clause is expressed as a binary variable. The computation is based on BDD[Bayes Decision Diagrams] - some improvements in this area improved ProbLog to be able to deal with quite large datasets. I must point out that recently the new version ProbLog2 was released, but there is only the web browser based input environment for this reasoner. The advantage of this reasoner is also the expressiveness given by the underlying Prolog language - it means we can construct some structures such as lists and can execute mathematical computations.

3.1.1 Computation of the probability

The computation of probability can be split in two parts; one approach is analytically precise and the second one is approximate for big queries which are suitable for the most of real life applications.

The problem of computation of the analytical solution for all the successful queries is transformed to compute the probability of DNF formulae. Even computing the probability of this representation is NP-hard task, so this is the situation when BDD[Bayes Decision Diagrams] are used. This process is still hard to solve, and it depends on the ordering of involved binary variables, but incorporating the same heuristics allows ProbLog to solve queries with more than tens of thousands of proofs.

This algorithm for solving approximate solution is based on iterative deepening. This approach is quite elegant and simple because the underling Prolog relies heavily on it. The SLD-tree can be searched very straightforwardly with iterative deepening; it also avoids the infinite branches in the tree. The iterative deepening can also be used in the process of computing the lower and the upper bound for the probability.

3.2 MLN

Other lightweight approach to combine FOL and probability is MLN — Markov Logic Networks. Inference in the MLNs is performed by MCMC (Markov Chain Monte Carlo) methods over the minimal subset of the network satisfying the query [16]. There is no other restriction than finiteness of the domain. Markov logic network is first-order logic knowledge base, where each formula is attached to its weight. This approach has several advantages, it even allows to infer even from a contradictory (in strictly logical sense) database. We can also take an advantage of the preprocessing of the input. The FOL-like syntax is parsed into C code. This can be added to the definition file containing function and used in the definitions; this enhances the expressiveness of the Alchemy input.

3.2.1 Computing the probability, inferring and learning

The problem is, as it was already stated, intractable in general. As in ProbLog there must exist a substitution with binary variables for each formulae and the resulting probability is simply the number of the ground literals after the Markov chain has converged. The probability of each first-order logic rule is corresponding to one (actually a series of grounded) clique in Markov networks; therefore, the following formula is to compute the satisfiability of the FOL formula from facts. In the following equation the n_i stands for the number of times the ith formula is satisfied by the state of the world x and Z is a normalization factor to make the probabilities sum to one.

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{3.2}$$

So this computation in the MLN is realised with Gibbs sampling. Since the Gibbs sampling algorithm is local minima search, the inference is started several times.

3.3 FuzzyDL

Since the problem can be solved by software mentioned above, there is another way to derive the certainty that such object can be classified as member of some class. It doesn't rely on the probability but simply answers the queries in a way of Fuzzy logic. It is an extension of standard Description logic. It is possible to derive some new facts from the knowledge base of DL, but all the result will be only a boolean value. Since this implementation involves Fuzzy logic, the answer for the queries are not only $\{0, 1\}$, but can receive some value from $\langle 0, 1\rangle$. We simple compute the confidence degree.

3.4 Reasoning examples

Let's consider following example — I have tried to express smoker's example in each reasoner. I have tried to keep them as similar as possible to provide similar and meaningful interpretation of results. Each of the examples contains a database containing persons. About some of them we know that they are smokers. There are relations between the persons and people can influence each other. Our goal is to determine the probability of someone to be a smoker from the person's relations.

3.4.1 MLN

The file that defines the word we would like to infer in is described in file `smokers.mln`

Listing 3.1: Alchemy MLN file

```
//predicate declarations
smoker(person)
friends(person,person)

//formulas

0.6 smoker(p1)^friends(p1,p2) => smoker(p2)
```

The file that contain all known facts (the database) is in `smokers-train.db`

Listing 3.2: Alchemy db file

```
smoker(Bob)

friends(Bob,Anna)

friends(Bob,Cloe)
friends(Anna,Cloe)

friends(Cloe,David)
```

And the last file we need to use the Alchemy software is file describing the role of each variable that can appear in the reasoning process are stored in `smokers-train.mln`

Listing 3.3: Alchemy variable description

```
person = { Anna, Bob, Cloe, David }
```

The reasoning in Alchemy is invoked by typing the command

Listing 3.4: Alchemy inferring

```
infer -i smokers.mln -e smokersTrain.db -r smokers.results -q smoker
```

What we get as a result can be seen in declared file. Its content is:

Listing 3.5: Alchemy results

```
smoker(Anna) 0.611989
smoker(Cloe) 0.70398
smoker(David) 0.576992
```

In this simple case it is relatively easy to interpret the results. We know that Bob is a smoker; therefore, is not included in the results. Anna is connected to the Bob, so there is a probability of being smoker about 0.6. Cloe is connected with Bob and Anna, the probability is higher because she is connected with more potential smokers. And the probability for David is slighter than the others because it can be derived only from not a certain connection with Cloe.

3.4.2 ProbLog

For reasoning in ProbLog we have to install the YAP and load a ProbLog library to use its predicates. The file for the following example looks like

Listing 3.6: ProbLog input

```
:- use_module(library(problog)).

0.6::smoker(bob).
0.4::smoker(cloe).

smokes(X) :- smoker(X).
smokes(X) :- friend(Y,X),smoker(Y).

friends(Y,X,[]) :- friend(Y,X).
friends(Y,X,[Z|R]) :- friend(Y,Z),friends(Z,X,R).

friend(bob,anna).
friend(anna,cloe).
friend(cloe,david).
friend(bob,david).
```

The ProbLog query is executed by typing into yap CLI, for example (and in this case we can allow it) we can ask for exact solution for David being a smoker.

Listing 3.7: ProbLog results

```
    ?− problog_exact(smokes(bob),Prob,_).
Prob = 0.6
    ?− problog_exact(smokes(anna),Prob,_).
Prob = 0.6
    ?− problog_exact(smokes(cloe),Prob,_).
Prob = 0.4
    ?− problog_exact(smokes(david),Prob,_).
Prob = 0.76
    ?− problog_exact(smokes(emilly),Prob,_).
Prob = 0.0
```

The clarification what this results means can be as follows. The queries that find an exact match, i.e. `bob` a `cloe` are clear, in the case of nonexistent person `emilly` we see zero probability; it also makes sense. It is because there is no connection among the persons in database and the person we are asking about. The probability for `smokes(anna)` is derived from the definition of the `smokes` predicate. The probability for `david` could be interpreted as the inverse of the case that both `cloe` and `bob` are not smokers, i.e. $1 - 0.6 * 0.4 = 0.76$.

3.4.3 FuzzyDL

The FuzzyDL input file is obviously different, the reasoner has completely another structure. We have the database and the query in one file and as a result we get the min and max instances inferred by the FuzzyDL reasoner.

Listing 3.8: FuzzyDL input

```
(implies (some friendOf Smoker) Smoker 0.6)
(symmetric friendOf)

(related anna bob friendOf)
(related bob cloe friendOf)
(related cloe david friendOf)
(instance anna Smoker)
(instance david (not Smoker) 0.8)

(min−instance? anna Smoker)
(min−instance? bob Smoker)
(min−instance? cloe Smoker)
(min−instance? david Smoker)
(max−instance? anna Smoker)
(max−instance? bob Smoker)
(max−instance? cloe Smoker)
(max−instance? david Smoker)
```

The results of the reasoner are as follows:

Listing 3.9: FuzzyDL results

```
Is anna instance of Smoker ? >= 1.0
Is bob instance of Smoker ? >= 0.6
Is cloe instance of Smoker ? >= 0.2
Is david instance of Smoker ? >= 0.0
Is anna instance of Smoker ? <= 1.0
Is bob instance of Smoker ? <= 1.0
Is cloe instance of Smoker ? <= 0.6
Is david instance of Smoker ? <= 0.2
```

3.5 Application examples

In this chapter I will provide a comparison of existing application examples where the introduced reasoners were used. It appeared that most of the examples are rather academic examples than real-life applications. The MLN has the most examples, the ProbLog only several, but for FuzzyDL I was unable to find a single one. I also haven't found any resource comparing these reasoners on one dataset or other direct comparison regarding either the results or either their performance.

3.6 ProbLog

3.6.1 Data acquisition and modeling for learning and reasoning in probabilistic logic environment

In this section I'm relying mostly on an article describing several different problems for ProbLog [17]. In the article authors propose several levels of abstraction for modelling the problem of resolving the probabilities. This is interesting because that is one of the major improvements – to select the level of abstraction for selected problem, from a level with no abstraction containing the full complexity of given task to a level with the highest one. The three layers among these extreme situations can be used to model a problem depending on its complexity and structure.

In the first case the famous novel *Les Misérables* by Victor Hugo is examined, the co-appearance of characters in chapters is investigated. In the beginning we can simply compute the probability for each connection between two characters in all chapters. This model is represented with a graph where the nodes are the characters and the edges represent co-appearance in each chapter. The edges are also labelled with number of chapters. If we enhance the model even just slightly, for example with the information that the novel consist of 356 chapters, we can derive much more interesting results. This enhancement can allow us to compute the probability of appearance for two characters in given chapter or even how many chapters is necessary to read to connect two characters together.

In the next example the, article describes creation of a probabilistic dictionary. We have a dictionary containing about 250 words and the meaning for only 30 of them. We also have connection for each word with a subset of the dictionary in relation of synonyms. In the end

we can compute the probability of connection between the word from the dictionary and the definition. Then we can probably obtain the meaning that we have in dictionary but it is not directly explained. This piece of information came from the connection synonym relation between this word and the others in the dictionary.

The last example is inspired by the nature. The authors of the work tried to model the neural network of a worm. They built a database of connections, of neurons and connection between them. In the end they created probabilistic graph of neurons and define edges as probabilistic signals, from this structure they were able to compute the probability for neuron being excited by certain transmitter.

3.6.2 Link and Node Prediction in Metabolic Networks

This article [12] describes the process of modelling the metabolic network into a probabilistic model. The task was to predict links and nodes. The ProbLog implementation achieved better results but it was not much significant, nevertheless it was capable to model the background knowledge even though it was necessary to involve uncertainty. The ProbLog representation of the network was used to predict nodes and connections. This approach showed an improvement over baseline solution especially when the data were more noisy. The experimental evidence shows that this application of ProbLog framework can partially recover missing information and correct inconsistent information of the metabolic network.

3.7 MLN

3.7.1 Jointly Modeling WSD and SRL with Markov Logic

This article [7] describes how the MLN could be used in processing the natural language. The proposition is to make a Markov logic model that jointly labels semantic roles and disambiguates all word senses. This approach could manage to solve both separate tasks without ignoring the logical constraints between them. The expressiveness of the MLN allowed to constrain the models and pipe-line them in several different configurations. The results have proved statistically significantly that the baseline solution which was using these system separately was improved.

3.7.2 Prediction of protein β-residue contacts

From this paper [13] we can conclude that MLN can be used for improving prediction of proteins interactions. This task is an example of a supervised link prediction; the MLN was used for creation of a model of conditional distribution of the connection between the proteins. The conclusion after recomputing the BetaPro[1] test example (916 protein chains and 48 996 β-residues) provides the information that slightly modified version of Alchemy scored about 8% percent better than the state of art solution while comparing results containing less then 10% percent of all proteins wrong.

[1] current state-of-art protein β-residue predictor

3.7.3 Goal Recognition for Player-Adaptive Games

The MLN (namely the Markov theBeast[2]) has proven in this work [11] usability even for real world tasks. There had been a corpus of observed actions (77182) from certain educational game. The researchers with the help of MLN built an undirected probabilistic graphical model. The parameters for the model were learnt from the corpus of actions. The task was to classify the most likely goal associated with the currently observed player actions. In the graphical model there are present observed states such as game state, previous actions and hidden state representing current goal. The actions leading directly to the goal were removed in order to ensure that the training model was fair. The decision system they used in comparison was a trivial model - predicting the goal appearing most frequently, and two non-trivial, one taking into account current action and the second one incorporating the previous as well. It outperformed the current system for determining the goal recognition actions and increased the correct predictions by approximately 82% comparing with the trivial classifier.

3.8 FuzzyDL

FuzzyDL reasoner can be used in task involving matchmaking; a simple example can be found in the introduction paper for the reasoner [6]. Some parts of problems that could be solved by FuzzyDL reasoner can also be transformed from Fuzzy OWL to standard OWL ontologies thanks to the plug-in for Portége ontology software. The authors claim that this conversion is useful and ready to use; at this point I have to express my concern about this quite strong opinion and it is worth to notice that there is no example using this technique. This reasoner could be also used for Fuzzy control systems, but I have not found any examples.

3.9 Conclusion about reasoners

After reading several articles I have concluded that the model that I'm planning to introduce and afterwards reason about in future should be as simple as possible due to the limitation of the reasoners. For example this paper [18] shows how could sometimes be a problem even to express the task in the framework or get the right results. The size of the domain would be a smaller problem than the way how to represent the data in a meaningful manner but still soluble. It is rather complicated to decide which approach is most convenient for certain task. I haven't found any paper comparing selected framework on one task and they can be hardly compared to more traditional ways of inferring either, because the tasks have mostly not been solved yet.

[2] http://code.google.com/p/thebeast/

Chapter 4

Model and learning

This chapter covers the process of extracting probabilistic features used afterwards for creating a probabilistic model describing the objects. Then the learning of parameters for probabilistic facts is described using methods of the chosen probabilistic reasoner, i.e. ProbLog.

4.1 Probabilistic features

The probabilistic fact for each object in the map can be encoded in several ways. The object could be described with the distribution function like `parcel(object,distribution function)`. This input would be available with FuzzyDL. But we decided not to use this reasoner due to its lack of applications. But unfortunately, from the reasoners left none is capable of handling this kind of input. So the only way left is to compute the probabilities in advance and store them as facts.

For creating the probabilistic features the data obtained from the processed image are used. For each object labelled on the map we know its description shown in table below. From these features the probabilistic ones are created.

processed features
size
size of its convex hull
coordinates for each 'corner'
number of neighbours
included objects

Table 4.1: Processed image features

computed features
circumference
vertical angle of corners
horizontal angel of corners

Table 4.2: Computed image features

Table 4.3: Image features

4.1.1 Size

There are three probabilistic facts derived from previously mentioned features (in table 4.3). The first one is the ratio of the size of the whole picture and selected label

$$size(x) = \frac{sizeOfObject(x)}{sizeOfPicture} \tag{4.1}$$

The next feature is computed as quotient after dividing the size of convex hull and the size of the object.

$$convexness(x) = \frac{sizeOfObject(x)}{sizeOfConvexHullPOfObject(x)} \tag{4.2}$$

The last feature is created only for better understanding what's happening during the

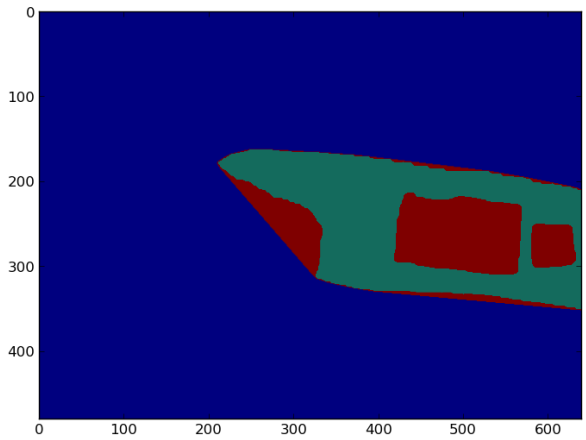


Figure 4.1: Filling the convex hull
The cyan colour stands for one selected object and the red colour represent its convex hull, in this case the resulting probability is 0.654

reasoning and it is defined as follows:

$$nonconvexness(x) = 1 - convexness(x) \tag{4.3}$$

The last feature is rather exotic. It is based on an idea how to detect the existence of long and narrow objects. For this type of object it is typical to have long circumference and quite a small volume. When we would like to encode it as a probabilistic fact, we have to transform the facts to get the resulting ration on interval $[0; 1]$. In extreme case the volume would be $1 \cdot lenghtOfObject(x)$ and its circumference would look like $2(1 + lenghtOfObject(x))$. From this assumption and to satisfy the rule that the maximal probability is 1 an equation can be formed.

$$spaghetiness(x) = \frac{circumferenceOfObject(x)}{2 \cdot sizeOfObject(x)} \tag{4.4}$$

4.1.2 Angles

The other features handling computed angles between the lines connecting the corners. There are two analogical features for horizontal and vertical case. The features tell us how close to parallel relation the line segments are. The computed angles are shown on picture 4.2.

$$parallelHorizontal(x) = \frac{angleBetweenLineSegmentsHorizontal(x)}{180} \tag{4.5}$$

$$parallelVertical(x) = \frac{angleBetweenLineSegmentsVertical(x)}{180} \tag{4.6}$$

The other computable fact is how much the object in the language of these fact resembles

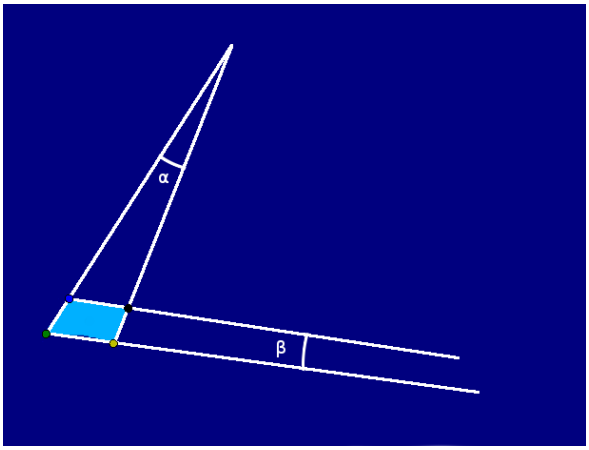


Figure 4.2: Computed angles
The angle β represent horizontal angle between the object's coordinates and the vertical is α, for this example the values are: horizontal angle is 179.0587° and vertical angle 171.4264° which yield facts parallelHorizontal=0.995,parallelVertical=0.952

square.

$$square(x) = \frac{\sqrt{sizeOfObject(x)}}{circumferenceOfObject(x)} \tag{4.7}$$

4.1.3 Relational data features

There is only one relational feature obtained from image description. It is a simply ratio of the number of neighbours of selected object to maximal number of neighbours in certain image. As in the previous example the negation of the feature is created.

$$NeighboursRatio(x) = \frac{countOfNeighbours(x)}{maximumOfNeigbours} \tag{4.8}$$

$$notNeighboursRatio(x) = 1 - NeighboursRatio(x) \tag{4.9}$$

For the other fact there was no suitable probability interpretation so they were left crisp. This features are namely 'is in', 'next to' and 'is center of mass in object'. The complete ProbLog output for one object look like code example below.

Listing 4.1: Probabilistic features

```
0.934::parallel_v(p5).
0.852::parallel_h(p5).
0.007::size(p5).
0.230::convex(p5).
0.770::nonconvex(p5).
0.018::spaghetiness(p5).
0.358::max_neigh(p5).
0.642::min_neigh(p5).
0.132::square(p5).
contain(p5,p99).
next_to(p5,p99).
next_to(p5,p119).
cm_in(p5).
```

4.2 Getting ProbLog working

For probabilistic reasoning there is a necessity to compile YAP and all with all the packages such as CUDD that ProbLog requires. The whole procedure how to do it is on official ProbLog web pages[1]. In order to tell the Prolog that we would like to use the probabilistic environment of ProbLog we have to start the Prolog code with such line:

Listing 4.2: ProbLog loading

```
use_module(library(problog)).
```

4.3 Model

The whole model consists of these definitions. They will be explained and justified why we have created these particular ones. The probabilistic data known for each object are shown in code excerpt number 4.1

4.3.1 Street model

For description of the street on the map we have decided to count into account features, that describes the shape of it and the number of neighbours. The typical street is either convex shaped with large number of neighbouring objects, regardless whether they are parcels or houses. When the street is curved, the neighbour relation remains unchanged but the shape is now nonconvex. The last case covers in our model represents the situation when the street is for example on the edge of the area or has parcels alongside. Than we assume that is nonconvex and has big spaghetiness.

[1] http://dtai.cs.kuleuven.be/problog/installation.html

Listing 4.3: Street model

```
street(S) :- max_neigh(S),nonconvex(S).
street(S) :- max_neigh(S),convex(S).
street(S) :- nonconvex(S),spaghetiness(S).
```

4.3.2 House model

Considering the house we assume that this kind of object is contained in another object; therefore, all the relations dealt with contain this crisp feature. The building is rather convex or square. It is necessary to separate these two descriptions because based on our features — the convexness does not guarantee the squareness. And the square relation can describe cases when the shape of house is more complex but the ration between circumference and volume of the object are similar to square.

Listing 4.4: House model

```
house(H) :- convex(H),contain(_,H).
house(H) :- square(H),contain(_,H).
```

4.3.3 Parcel model

The last object category is a parcel. There is an assumption that parcels are rather big so the feature `size` is used.

Listing 4.5: Parcel model

```
parcel(P) :- convex(P),min_neigh(P).
parcel(P) :- contain(P,_).
parcel(P) :- convex(P),parallel(P),size(P).
```

4.3.4 Supporting code

There have been implemented several supporting Prolog predicates used in reasoning. The `parallel` predicate handles both vertical and horizontal parallelism. The maximum function is for deciding which probability outputted from the reasoner is the highest, and therefore which category the object most probably belongs to.

Listing 4.6: Supporting predicates

```
parallel(O) :- parallel_v(O).
parallel(O) :- parallel_h(O).

maximum(P1,P2,P3,P1,house) :- P1 >= P2,P1 >= P3,!.
maximum(P1,P2,P3,P2,street) :- P2 >= P1,P2 >= P3,!.
maximum(_,_,P3,P3,parcel).
```

4.4 Relational model

We also have tried to create a relational model that will contain the relational feature between objects. The preceding model was extended and contained recursive predicates. We have defined symmetric relation based on the `next_to` predicate.

Listing 4.7: Neighbour relation

```
neighbours(P1,P2) :- next_to(P1,P2).
neighbours(P1,P2) :- next_to(P2,P1).
```

Definition of each object on the map has been enhanced for its immediate neighbourhood. To limit the level of recursion, the nonrecursive predicate has been used. The nonrecursive predicate has the same definition as the original object except the relational. It is shown on example below:

Listing 4.8: Relational house model

```
house_n(H) :- convex(H),contain(_,H).
house_n(H) :- square(H),contain(_,H).

house(H) :- convex(H),contain(_,H).
house(H) :- square(H),contain(_,H).
house(H) :- neighbours(H,H2),neighbours(H,H1),
            house_n(H2),house_n(H1),
            neighbours(H,S),S \= H1, S \= H2,street_n(S).
```

But unfortunately even this simplest relational model has proved to be uncomputable on the real dataset. We have the toy model with only 20 objects. With the toy dataset the relational model worked flawlessly, with no significant time difference to the original model. But for the real datasets 8GB of RAM was not sufficient amount to perform the reasoning.

4.5 Learning

For learning the parameters in the model we have used ProbLog feature called `Parameter Learning From Partial Interpretations`. The process of implementation follows the tutorial on the official web page of ProbLog[2]. We have to rewrite all predicates in form that the ProbLog can interpret them as causes used in the learning process.

We have added to the clauses of the previous model predicates that will be learnt:

Listing 4.9: Terms to learn

```
t(_)::house1.
t(_)::parcel1.
```

The `t` is an abbreviation of `tunable` parameter. The `(_)` informs ProbLog that the initial probability is unknown and has to be learned. The `house1` is only the identifier of the term. The predicate enriched by the probability would look as follows:

[2] http://dtai.cs.kuleuven.be/problog/tutorial-learning-lfi.html

Listing 4.10: House learn model

```
house(H) :- house1,square(H),contain(_,H)
```

But for informing ProbLog that there is a clause to learn we have to encapsulate it into predicate `myclause/2`. The syntax of this clause is:

Listing 4.11: Clauses for learning

```
myclause(street(S),(street3,nonconvex(S),spaghetiness(S))).
myclause(parallel(O),(parallel_v(O))).
myclause(parallel(O),(parallel_h(O))).
```

Note that is necessary to encode the predicates that do not include probabilistic facts to get the learning working.

For the learning it is necessary to encode all the known facts. The data are generated from the classified dataset. The classification file is loaded into set. Then all the possible combinations are created by the Python set constructor. Then the resulting fact is a difference of these two sets:

Listing 4.12: All possible classifications

```
predicates=["house","parcel","street"]
examples=range(1,number_of_examples)
values=["true","false"]
set = {(o,n,v) for o in predicates for n in examples for v in values}
```

The resulting set has to be encoded following the rules so as ProbLog can understand it. Because our classifier can distinguish three categories we have one positive and two negative examples per object.

All the examples have to be introduced by clause:

Listing 4.13: ProbLog example

```
example(1). % 1 stands for example identifier
```

Listing 4.14: ProbLog knowledge base

```
known(1542,house(p1542),false).
known(1542,parcel(p1542),true).
known(1542,street(p1542),false).
```

The first number in the clause points to the `example` identifier, the second argument is the predicate and third argument is its truth value. In this case we encode both positive and negative examples. This is important to encode as much knowledge as we have, because the resulting learned probabilities differ a lot. This means not only to encode all the positive examples, but state the other possibilities as negative examples. There was a problem with positive examples about `house`, from some reason the learning algorithm was not able to satisfy the clause. Our assumption is that this could be caused by the crisp relation in the

definition in the house model. We have also excluded the small objects considered as ignored from the learning process.

The parameter in the code 4.15 refers to the number of iteration executed by the learning algorithm.

Listing 4.15: ProbLog learning

```
do_learning(10).
```

4.6 Results of classification

Let's take a closer look to the results of reasoning. The result is taken out of comparison of the computed probabilities. The object is assigned to the category with the highest probability. At first there is a table with information about the reasoning and the learning process. From there it is obvious that with the number of objects in the picture, the time of reasoning and also learning grows linearly. For the dataset `most` we were not able to perform

dataset	number of objects	time of classification	time of learning
bozi_dar	2072	1:51.82 s	20:03.28 s
brno_centrum	2189	2:25.35 s	20:28.36 s
jizni_mesto	2337	2:23.69 s	23:54.49 s
most	2473	2:31.41 s	7:14.69 ERROR
praha_centrum	2527	2:31.23 s	20:00.46 s
slapanice	3814	3.59.82 s	49:45.86 s

Table 4.4: Time of computation on datasets

the whole learning process, ProbLog during the computation of the probabilities came to resulting probability `nan`, and therefore could not proceed to the next iteration. Because it happened on one dataset only we decided to ignore this faulty result and used this dataset as a cross-validation dataset only.

We have tested whether the results of different type of dataset are different. The table below shows that against our expectation, the results are quite similar. The biggest difference can be found in the `pracel1` term. Surprisingly it does not matter what kind of dataset we compare.

Now we can examine how much the results differ during the learning process. As we can see in table 4.6, the results of certain terms differs a lot. But when we reasoned with the results after tenth iteration and the one hundredth iteration, the number of correctly classified object was the same. The difference between iterations can be seen in the table 4.6 for the dataset `Boží dar`.

In the end, let's compare the classification of the reasoner with the basic model without learning. As we can see the percentage of correctly classified object is rather low, at best the reasoner had reached about 50 % of correct results. The results for every dataset are in table 4.7.

term \ dataset	**bozi dar** countryside	**brno centrum** city centre	**jizni mesto** microdistrict	**most** microdistrict	**praha centrum** city centre	**slapanice** countryside
street1	0.308	0.372	0.310	∅	0.301	0.299
street2	0.521	0.542	0.473	∅	0.482	0.554
street3	0.416	0.406	0.345	∅	0.402	0.418
house1	0.703	0.703	0.703	∅	0.703	0.703
house2	0.758	0.758	0.754	∅	0.758	0.758
parcel1	0.929	0.939	0.890	∅	0.835	0.695
parcel2	0.852	0.852	0.852	∅	0.852	0.852
parcel3	0.540	0.540	0.540	∅	0.541	0.534

Table 4.5: Computed predicates on datasets

term \ iterations	1	10	20	50	100
street1	0.288	0.308	0.331	0.400	0.514
street2	0.518	0.521	0.524	0.531	0.541
street3	0.424	0.413	0.400	0.361	0.298
house1	0.703	0.703	0.703	0.703	0.703
house2	0.758	0.758	0.758	0.758	0.758
parcel1	0.876	0.926	0.929	0.929	0.929
parcel2	0.852	0.852	0.852	0.852	0.852
parcel3	0.539	0.540	0.540	0.540	0.540

Table 4.6: Comparison of iterations

dataset	number of objects	correct	errors	skipped	**percentage**
bozi_dar	2072	452	433	1188	51.13
brno_centrum	2189	647	1232	311	34.45
jizni_mesto	2337	673	812	853	45.35
most	2473	756	1432	286	34.57
praha_centrum	2527	549	1402	577	28.15
slapanice	3814	810	1935	1070	29.52

Table 4.7: Reasoning results

But if we reason with the learned predicates, the results are obviously much better. The best result is now almost 85 % of correctly classified objects. All of the results are at least above 50 % correct. The test results are taken from the learned probabilities after tenth iteration, the results between the datasets can be caused by the different count of objects or different structure of the data. The results are in table 4.8.

As we have mentioned, the datasets were separated in three categories. In the cross-validation we have used the learned probabilities on the first example of certain kind of dataset and used them for classification on the second; i.e. learned probabilities on `praha_centrum` was used on `brno_centrum` and vice versa.

These results do not correspond to what we expected. We expected increase of the num-

dataset	number of objects	correct	errors	skipped	**percentage**
bozi_dar	2072	745	140	1188	84.28
brno_centrum	2189	1323	556	311	70.45
jizni_mesto	2337	1124	361	853	75.74
most	2473	1343	845	286	61.41
praha_centrum	2527	1354	597	577	69.44
slapanice	3814	1673	1072	1070	60.97

Table 4.8: Results of learned reasoner

ber of correctly classified object on the original dataset and decrease on the cross-validation dataset due to overfitting. But the results regarding the percentage do not differ a lot with the number of iterations of learning steps even on cross-validation dataset. Therefore, we suggest that for similar sized datasets the 10 steps of learning are sufficient for classification. The bigger the dataset is, the higher value of steps we recommend to perform.

The figures below show how the learning process performs during the learning. The first subgraph informs us how the learned term changes with the increasing number of iterations of learning algorithm. As we can see, some of them change their values with every iteration and seem to converge to some value.

The graph below is a computed probability of given interpretation; this probability was computed according to the equation provided in this article [10].

$$\hat{\mathbf{p}} = \arg\max_{\mathbf{p}} P(\mathcal{D}|\mathcal{T}(\mathbf{p})) = \arg\max_{\mathbf{p}} \prod_{m=1}^{M} P_w(I_m|\mathcal{T}\mathbf{p})) \tag{4.10}$$

Where $\mathcal{T}$ is given ProbLog program with probabilistic facts $\mathbf{p}$. The $\mathcal{D}$ stands for set of possibly partial interpretation I_m. We are finding the maximum likelihood probabilities $\hat{p}$.

This criteria is used for the internal learning algorithm and we were interested in it only for the purpose of examining the risk of overfitting. The improvement of this criteria is most visible on the first graph; on the others the resulting probabilities are so small, that we can compare only numerical values without having the trend visible in the graph.

The last graph shows how good the learned reasoner performed on original data and on the cross validation dataset. As we can see even if the percentage of correctly classified object on the original map remains almost the same, the algorithm doesn't overfit because the reasoning on the cross validation dataset yields the better results the higher iteration we have.

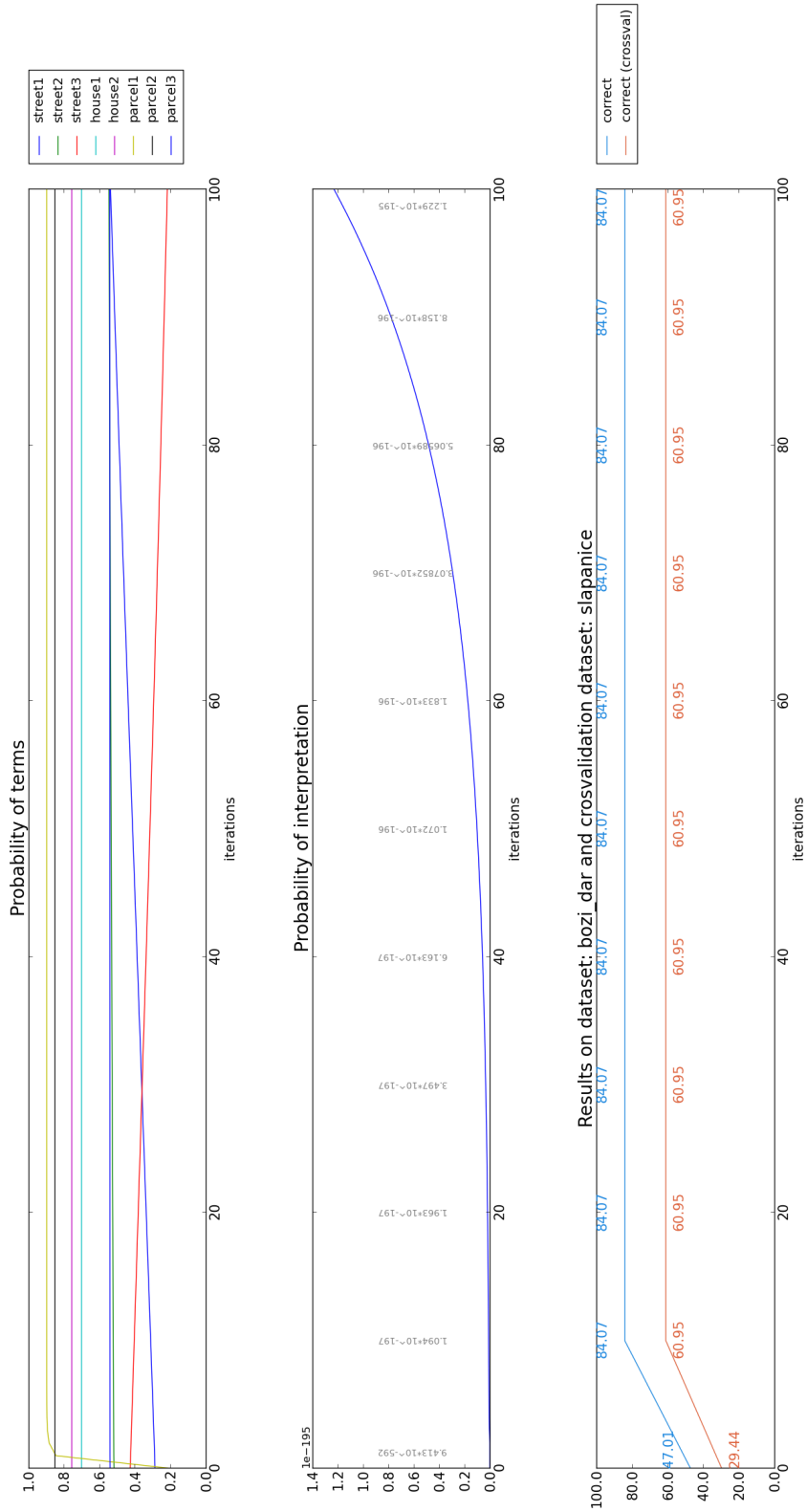


Figure 4.3: Bozi dar results

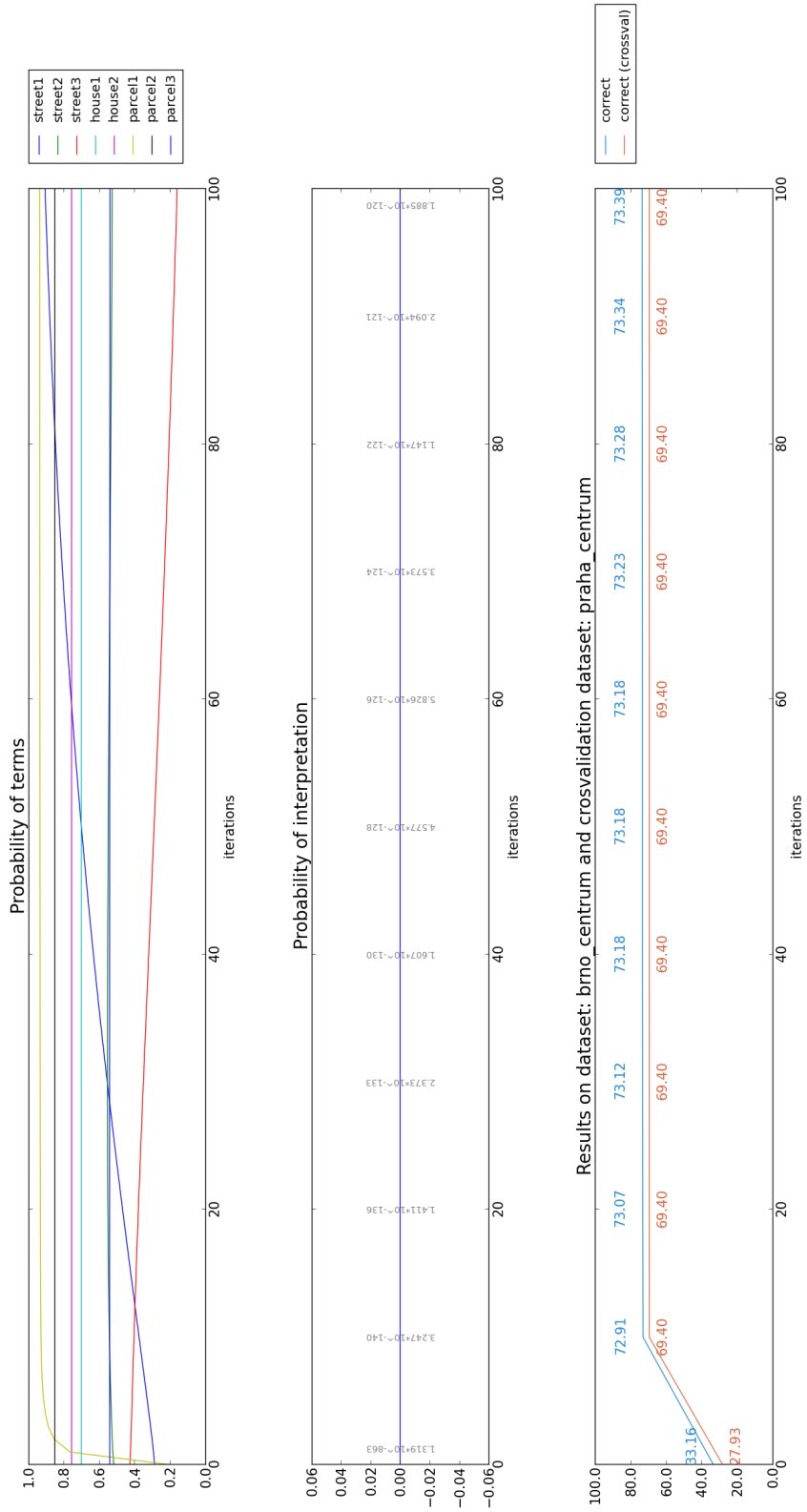


Figure 4.4: Brno centrum results

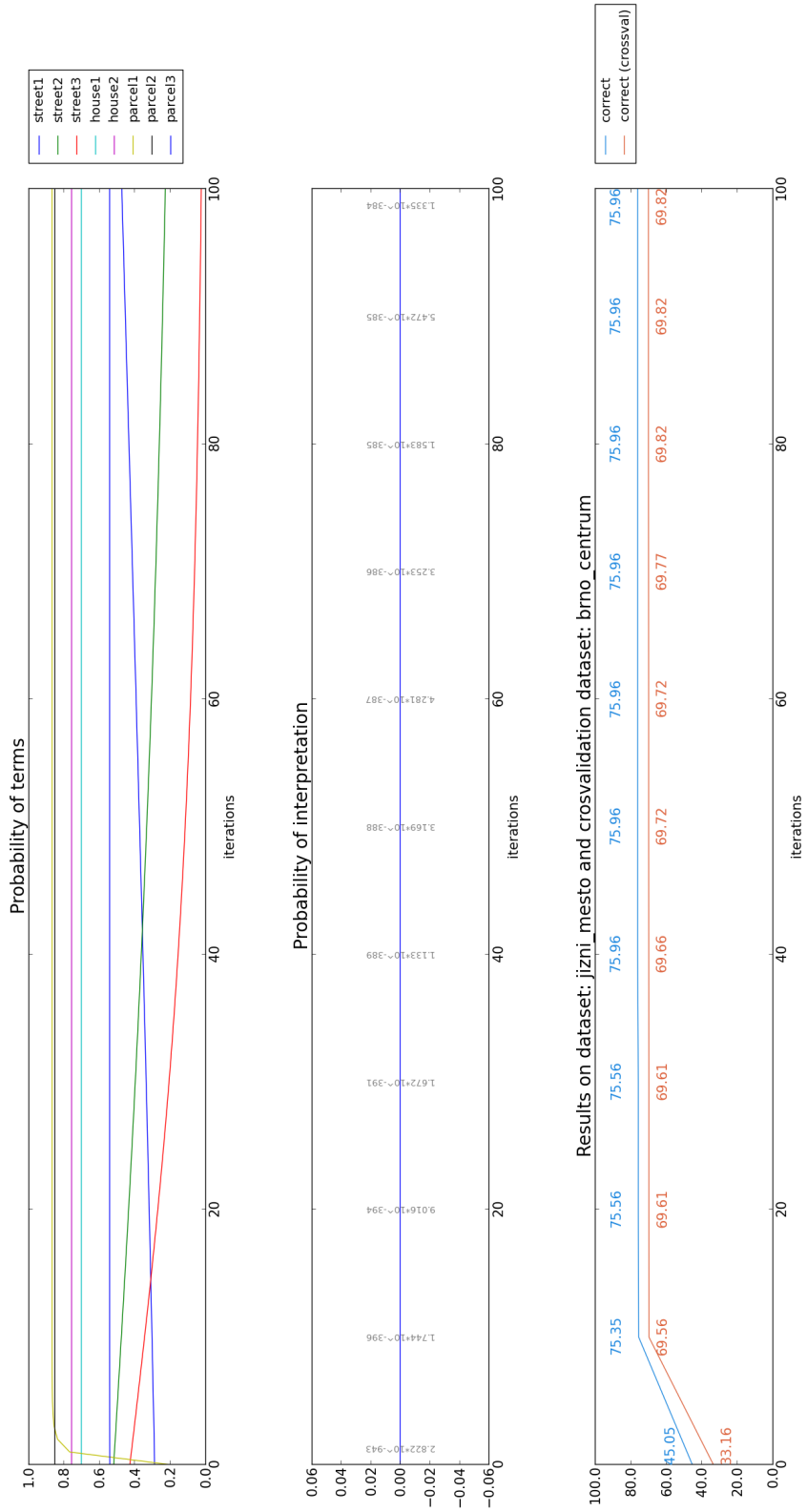


Figure 4.5: Jizni Mesto results

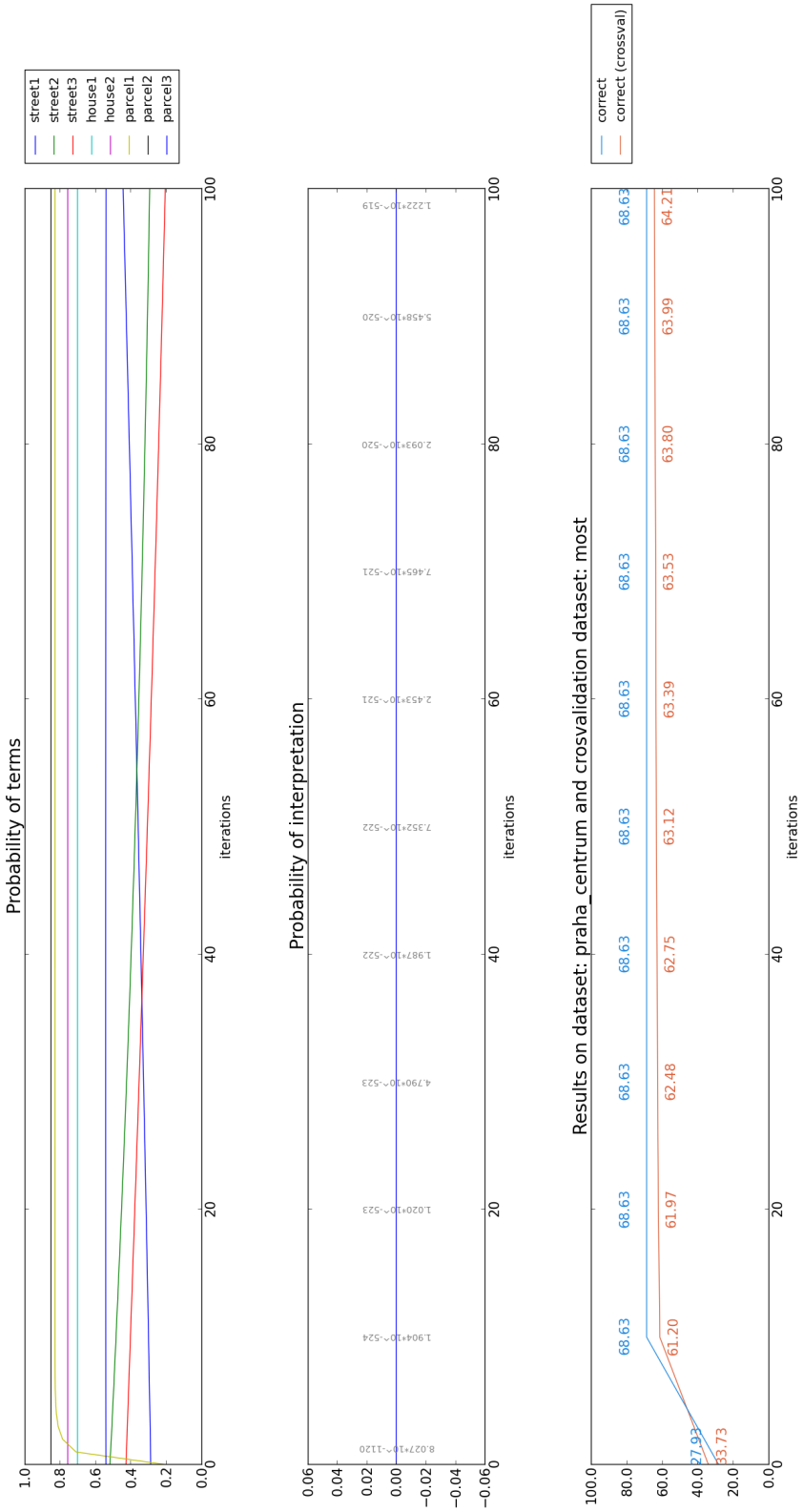


Figure 4.6: Praha centrum results

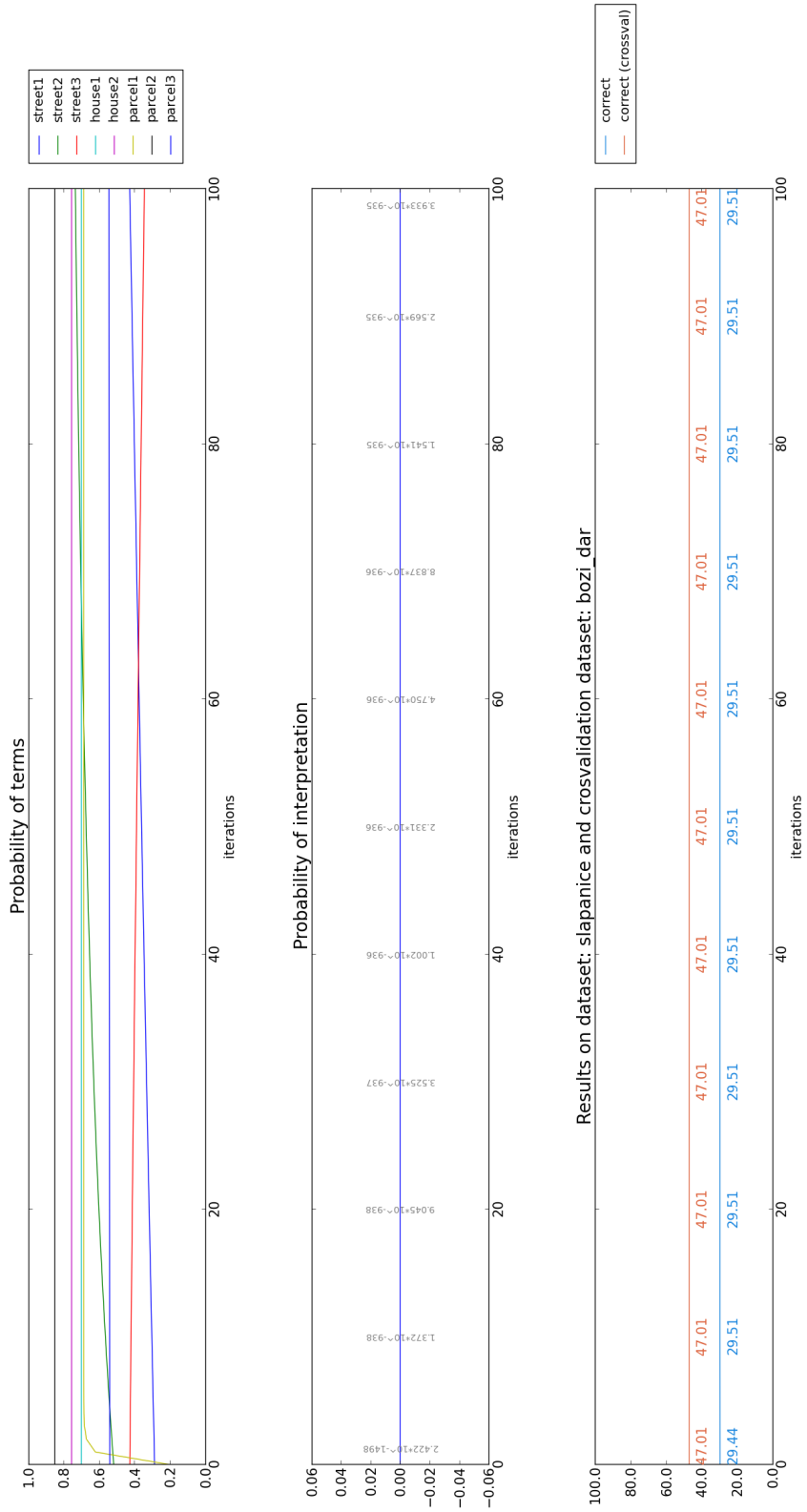


Figure 4.7: Slapanice results

Chapter 5

Conclusion and future work

As we have mentioned in the introduction, the aim of this work was to compare the state of the art probabilistic reasoners; process real world data; and test the quality of the reasoner.

We have created a basic model for reasoning and then tried several approaches how to improve it. From the results of reasoning we can conclude that the basic model without learning to describe independent features performs with unsatisfactory results. Therefore we tried two improvements of the basic model.

The first was employing the relational model that took into account relations among objects in the map. But unfortunately this model fails on the real world data due to its size. The other way how to improve it was to incorporate integrated learning algorithms within the ProbLog reasoner. The results have shown that even a few steps of learning algorithm can improve the accuracy of the model twice as much. This observation can be exploited in future when creating more sophisticated and complex models.

There are several ways to continue this work. For example the original model itself can be improved with ILP techniques to discover better models. The improvement can be based on adding new definitions for objects or on a try to create another features that can be obtained from the image for better description.

Other possibility for improvement is building a relational model that takes into account the relation between objects in the map during the reasoning. A simple example of this model was tested during our work, but it worked only on small sample dataset. On the real world dataset the reasoner was not able to use less then 8 GB of RAM; therefore, the computation could not been finished.

In the end we realize that the results are not good enough to use it in practice for the real applications. But the intention of this work was not concerned about this straightforward application. Our aim was to examine the usefulness of the probabilistic model and the relational features, for the classification. An in the end, to explore the ways how to improve this model.

Bibliography

[1] *Numpy* [online]. Available from: http://www.numpy.org.

[2] *Python Image Library (PIL)* [online]. Available from: http://www.pythonware.com/products/pil/.

[3] *Python, programing language* [online]. Available from: http://en.wikipedia.org/wiki/Python_(programming_language).

[4] *Scipy* [online]. Available from: http://www.scipy.org.

[5] *Web Map Service (WMS)* [online]. Available from: http://en.wikipedia.org/wiki/Web_Map_Service.

[6] BOBILLO, F. – STRACCIA, U. fuzzyDL: An expressive fuzzy description logic reasoner. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on*, s. 923–930. IEEE, 2008.

[7] CHE, W. – LIU, T. Jointly modeling WSD and SRL with markov logic. In *Proceedings of the 23rd International Conference on Computational Linguistics*, s. 161–169. Association for Computational Linguistics, 2010.

[8] COELHO, L. P. *Mahotas: Open source software for scriptable computer vision* [online]. January 2013. Available from: http://arxiv.org/abs/1211.4907v2.

[9] DUTCH, S. *Convert Between Geographic and UTM Coordinates* [online]. 2010. Available from: http://www.uwgb.edu/dutchs/UsefulData/ConvertUTMNoOZ.HTM.

[10] GUTMANN, B. et al. Parameter Learning in Probabilistic Databases. *Machine Learning and Knowledge Discovery in Databases*. 2008, s. 473. Available from: http://www.springerlink.com/index/10.1007/978-3-540-87479-9_49.

[11] HA, E. et al. Goal Recognition with Markov Logic Networks for Player-Adaptive Games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[12] KIMMIG, A. – COSTA, F. Link and node prediction in metabolic networks with probabilistic logic. *Bisociative Knowledge Discovery*. 2012, s. 407–426.

[13] LIPPI, M. – FRASCONI, P. Prediction of protein β-residue contacts by Markov logic networks with grounding-specific weights. *Bioinformatics*. 2009, 25, 18, s. 2326–2333.

[14] OTSU, N. A Threshold Selection Method from Gray-Level Histograms. *Systems, Man and Cybernetics, IEEE Transactions on.* Jan., 9, 1, s. 62–66. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

[15] RAEDT, L. D. et al. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, India*, s. 2462–2467, 2007.

[16] RICHARDSON, M. – DOMINGOS, P. Markov logic networks. *Machine Learning.* 2006, 62, s. 107–136. ISSN 0885-6125. doi: 10.1007/s10994-006-5833-1.

[17] SHTERIONOV, D. – JANSSENS, G. Data acquisition and modeling for learning and reasoning in probabilistic logic environment. In *Proceedings of 15th Portuguese Conference on Artificial Intelligence (EPIA 2011), Lisbon, Portugal*, 2011.

[18] TAGHIPOUR, N. – FIERENS, D. – BLOCKEEL, H. Probabilistic logical learning for biclustering: A case study with surprising results. *CW Reports.* 2010.

Appendix A

Content of attached DVD

The structure of attached DVD is as follows:

Listing A.1: Content of attached DVD

```
+-- code                                  ->      Containing the code
|   +-- trnkato2_BP.py                    ->      Main module
|   +-- utils                             ->      Utility module
|       +-- czuk_utils.py
|       +-- __init__.py
|       +-- map_utils.py
+-- docs                                  ->      Basic documentation for each file
|   +-- czuk_utils.html
|   +-- map_utils.html
|   +-- trnkato2_BP.html
+-- pictures                              ->      Datasets
|   +-- bozi_dar
|   |   +-                                ->      Folder containing data about image
|   +-- bozi_dar.jpg
|   +-- brno_centrum
|   |   +-                                ->      Folder containing data about image
|   +-- brno_centrum.jpg
|   +-- jizni_mesto
|   |   +-                                ->      Folder containing data about image
|   +-- jizni_mesto.jpg
|   +-- learn                             ->      Template for ProbLog learning
|   +-- mapa
|   |   +-                                ->      Folder containing data about image
|   +-- mapa.jpg
|   +-- most
|   |   +-                                ->      Folder containing data about image
|   +-- most.jpg
|   +-- praha_centrum.jpg
|   +-- praha_centrum
|   |   +-                                ->      Folder containing data about image
|   +-- praha_centrum.jpg
|   +-- reason                            ->      Template for ProbLog reasoning
|   +-- slapanice
|   |   +-                                ->      Folder containing data about image
|   +-- slapanice.jpg
+-- text
    +-- trnkato2_BP_2013.pdf              ->      This thesis
```