UNIVERSITY OF LEIPZIG

BACHELOR THESIS

Active Learning of Link Specifications using Decision Tree Learning

Author:
Daniel OBRACZKA

Supervisor:
Dr. Axel-Cyrille NGONGA NGOMO

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Faculty of Mathematics and Computer Science

January 2, 2017

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Structure . . . 2

2 Semantic Web **3**
2.1 Vision . . . 3
2.2 RDF & Ontologies . . . 3
2.2.1 Syntax . . . 4
2.2.2 RDFS . . . 4
2.3 Link Discovery . . . 5
2.3.1 Definitions . . . 5
2.3.2 Frameworks . . . 6
2.3.3 Machine Learning . . . 7

3 Decision Trees **8**
3.1 General Concept . . . 8
3.1.1 Goals . . . 8
3.1.2 Implementations . . . 9
3.2 Splitting a Node . . . 9
3.3 Measures of Purity . . . 10
3.3.1 Information Gain . . . 10
3.3.2 Gain Ratio . . . 12
3.4 Missing Values . . . 12
3.5 Pruning . . . 13
3.5.1 Pre-Pruning . . . 13
3.5.2 Post-Pruning . . . 13
3.6 Error Rates . . . 13
3.7 Further Refinements . . . 16

4 Approach **17**
4.1 Overview . . . 17
4.2 Creating Training Instances . . . 18
4.3 Learning . . . 19
4.4 Parsing the Decision Tree into a Metric . . . 20
4.4.1 General procedure . . . 20
4.4.2 Examples . . . 20
4.5 Check if there was a better link specification before . . . 21
4.6 Most Informative Link Candidates . . . 22

5 Experiments and Results **24**
5.1 Experimental Setup . . . 24
5.1.1 Measures . . . 24
5.1.2 Parameters . . . 24
5.2 Active & Batch Learning Experiment . . . 25

5.3 Active & Batch Learning Experiment Results 26
5.4 User Feedback Experiment 26
5.5 User Feedback Experiment Results 26
5.6 Experiments Summary 26

6 Conclusion and Future Work **31**

Bibliography **32**

Chapter 1

Introduction

The World Wide Web is enormous and evergrowing. In one second on the internet[1]

- 7411 tweets are sent,
- 751 Instagram photos are uploaded,
- 1179 tumblr posts are made,
- 39590 GB of internet traffic is transferred and
- 57256 Google searches are carried out.

Similarly, the Semantic Web is growing: there are over 2.5 billion Web pages that have markup according to `schema.org` format,[2] linked data is used by big media sites such as BBC and New York Times. Google, Yahoo!, Microsoft, Facebook and many other global players of the Web business are developing large knowledge graphs, defining, structuring and linking hundreds of millions of entitities. DBpedia [2] has grown from 103 million triples (DBpedia 2.0) in 2007 to 9.5 billion triples in 2016 (DBpedia 2016-04).

1.1 Motivation

While substantial effort has already been invested into making Tim Berners-Lee's vision of a shift from the document-oriented web towards a Web of interlinked data [5], there is still much work to be done. For example only 7.79% of the datasets in the LOD Cloud link to more than ten other datasets.[3] Linking data is essential for the success of the semantic web. However the task of linking knowledge bases can be tedious, especially if large datasets are used. In some cases, expert knowledge is needed to be successfull at this endeavour. Furthermore, knowledge bases underlie constant changes (e.g. adding of new instances), which calls for a machine backed linking process even more.
To address the aforementioned problems, we rely on previous efforts condensed in the LIMES [23] framework. This framework tackles the potentially quadratic complexity of linking datasets by using set theory combined with efficient planning algorithms. Therefore the approach of this

[1]Statistics taken from `http://www.internetlivestats.com/one-second/`
[2]`http://www.dataversity.net/schema-org-fires-lit/`
[3]`http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/#toc4`

work solely has to deal with the problem of finding appropriate similarity measures on properties of datasets to find links. We do this by using a seasoned machine learning technique: decision trees. Supervised learning techniques often require a great amount of training data. In our approach, we tackle this training data problem by implementing an active learning mechanism. By these means, we can lower the amount of user input by deciding smartly which data should be labeled.

1.2 Structure

This work is structured as follows: We start by giving an overview of the semantic web and decision trees in chapters 2 and 3. For each topic, we take a look at the general idea that lies beneath it and continue by diving into techniques used to implement this objective. Concurrently, we also give an overview of the related work in the respective field. Chapter 4 describes our implementation and how it makes use of the aforementioned concepts. The subsequent chapter 5 is dedicated to the presentation of results. We evaluated our approach on nine datasets (synthetic and real-world) and compared it with three state-of-the-art classifiers. Our approach outperforms the state-of-the-art on four of the nine datasets by up to 30% in regards to the F-Score, while still being time-efficient. However, the state of the art still provides better results on average. Ultimately, we conclude and give a prospect of future work.

Chapter 2

Semantic Web

2.1 Vision

The World Wide Web we use today is a colossal amount of documents that are primarily designed for human consumption. Although machines can manipulate the data, they primarily regard these documents as streams of characters. If machines just see character streams, only for humans the string “`bike`” is known as a vehicle that can be used to transport a person from A to B, usually with some amount of physical exercise as a byproduct. The Semantic Web aims to overcome this restriction by introducing certain standards on how information should be described. This makes it possible to share information between systems and platforms and even infer implicit facts. Obviously these standards have to be flexible and expandable enough to incorporate future changes. To tackle these problems, the *World Wide Web Consortium*[4](W3C) developed a number of standards including *RDF(S)*, *OWL* and *XML*. While XML is already in wide usage on the Document Web[5] RDF(S) and OWL are dedicated *ontology* languages for usage in the Semantic Web. An ontology is “an explicit specification of a conceptualization” [8, p.1]. In our case this specification is a set of types, properties and relationship types.

2.2 RDF & Ontologies

RDF is short for *Resource Description Framework* [16] and is a formal language to describe structured data. In contrast to XML, which uses a tree-structure, this language uses directed graphs. There are several reasons for this: While trees are a good way of storing information of documents in a strict hierarchical order, which can be easily searched and processed, RDF was invented to describe relationships between resources. These do not necessarily have a hierarchy. Another reason is that this language is designed for decentralized structures like the WWW and therefore new information has to be easily added, which is much harder in tree structures than in graphs.
An RDF graph consists of edges and nodes that have unambiguous identifiers called *URIs* (Uniform Resource Identifiers). The usage of URIs is necessary to avoid name clashes, which could lead to major inconsistencies in ontologies. For example the string “`Mercury`” could either mean a chemical element, a planet or a roman god, which are obviously very different things. A small example for a RDF graph can be seen in Figure 2.1. As we

[4]`http://www.w3.org`
[5]We refer to the classical Web2.0 as Document Web

can see the nodes have names that resemble the structure of *URLs* (Uniform Resource Locators). The reason for this, is that URIs are just a generalization of URLs. Another important aspect is the representation of data values

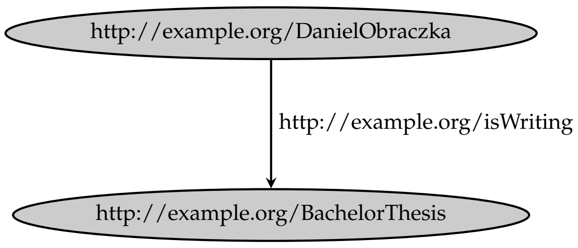

http://example.org/DanielObraczka
http://example.org/isWriting
http://example.org/BachelorThesis

Figure 2.1: *example RDF graph*

as *literals*. Literals can either have a annotated data type (in this case they are *typed*) or not (*untyped*), in which case they are treated as strings. Literals can never have outgoing edges, which means statements about literals are not permitted [11, pp.36-39].

2.2.1 Syntax

While the representation of relationships between resources as graphs is clear for small examples, it becomes confusing quite fast. Moreover, it would be very difficult to use this graphical representation for computational processing. The solution for this is to describe knowledge graphs as triples of *subject - predicate - object*. There have been several proposals on how to do this: one of the first was Tim Berners-Lee's *Notation 3* (N3) [4] in 1998. Since it also included more complex expressions, a subset of it called *N-Triples*[6] was devised, which was later expanded again to *Turtle* [3]. The latter is the unofficial standard for RDF-Syntax since it is expressive enough for most use cases, but very user-friendly [11, pp.40f].
If we want to translate Figure 2.1 into turtle it looks like this:

```
@prefix ex: <http://example.org/> .

ex:DanielObraczka ex:isWriting ex:BachelorThesis .
```

URIs are put into angle brackets and we used prefixes in this case to shorten them. This is done by declaring prefixes similar to namespaces in XML. So instead of writing `http://example.org/` everywhere we define `ex` as a contraction. RDF also has the possibility to declare datatypes or language of literals, use multivalent relationships, blank nodes or lists and much more we cannot describe here, but can be found in [11].

2.2.2 RDFS

So far we have explicated how RDF can be used to make statements about single resources. But we also want to represent classes with similar characteristics. This is were RDF Schema (RDFS) comes into play. It is a part of the W3C Recommendation for RDF and provides the basic elements for the description of (lightweight) ontologies. If we want to say something is an instance of a class we use the `rdf:type` predicate. E.g.:

[6]`https://www.w3.org/TR/n-triples/` This link points to the latest version, the original was proposed in 2004

```
ex:DanielObraczka rdf:type ex:Student .
```

tells us that `ex:DanielObraczka` belongs to a class `ex:Student`. To know that `ex:Student` is a class and not just another entity we use another predefined URI:

```
ex:Student rdf:type rdfs:Class .
```

Since some classes have also very similar characteristics, it is possible to declare class hierarchies. For example if we want to say that every student is also a person:

```
ex:Student rdfs:subClassOf ex:Person .
```

Of course we not only want to make statements about classes, but also about properties. For this we use `rdf:Property`. For example:

```
ex:isWriting rdf:type rdf:Property .
```

Similar to subclasses it is also possible to declare subproperties:

```
ex:isWritingSuccessfully rdf:subPropertyOf ex:isWriting .
```

Another important aspect for ontology engineering is the ability to restrict the types of resources a property can be used with. E.g. we don't want triples like

```
ex:DanielObraczka ex:isWriting ex:Car .
```

in our knowledge base. To ensure the right usage of properties we use `rdfs:domain` to restrict the possible types of subjects and `rdfs:range` to do the same for objects.

```
ex:isWriting rdfs:domain ex:Person .
ex:isWriting rdfs:range ex:Document .
```

So for our example we also have to declare that

```
ex:BachelorThesis rdfs:subClassOf ex:Document .
```

We have now discussed the basic tools RDFS gives us to model knowledge bases. Of course there is a vast number of expressive possibilities, like open lists, statements about statements (reification) etc. we could not touch in this work [11, pp.66-86]. Since RDFS lacks certain expressions such as e.g. negation OWL [7] is used for more complex ontologies.

2.3 Link Discovery

Ensuing, we will have a look at link discovery. We start by defining the problem and continue with the presentation of a few frameworks that try to tackle it.

2.3.1 Definitions

The link discovery problem, which shows a certain similarity to the record linkage problem,[8] is difficult to model formally, since it is an ill-defined problem [1]. In general, link discovery frameworks aim to compute the set

[7] https://www.w3.org/TR/owl-ref/

[8] "Record linkage is the process of identifying pairs of records that refer to the same thing" [39]

$M = (s,t) \in S \times T : R(s,t)$ where S and T are sets of RDF resources and R is a binary relation.

Definition 1 (Link Discovery) *"Given two sets S (source) and T (target) of entities, compute the set $\mathcal{M}$ of pairs of instances $(s,t) \in S \times T$ such that $R(s,t)$" [25, p.4].*

Usually the following applies: The set S is a subset of the instances contained in the knowledge base $\mathcal{K}_S$. This applies similarly to T and $\mathcal{K}_T$. Note, that neither S and T, nor $\mathcal{K}_S$ and $\mathcal{K}_T$, have to be necessarily disjoint. The question whether $R(s,t)$ holds for $s \in S$, $t \in T$ is carried out by comparing their properties via a (complex) similarity measure σ. If the value of $\sigma(s,t)$ is bigger or equal to a threshold θ, two instances $s \in S$ and $t \in T$ are considered to be linked via R. The details of this similarity condition and sets S and T are usually defined in a *link specification* [25].

Definition 2 (Link Specification) *"A link specification consists of three parts: (1) two sets of restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$ resp. $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$ that specify the sets S resp. T, (2) a specification of a complex similarity metric σ via the combination of several atomic similarity measures σ_1, ..., σ_n and (3) a set of thresholds τ_1, ..., τ_n such that τ_i is the threshold for σ_i" [24, p.3].*

Since we are dealing with RDF resources, a restriction $\mathcal{R}$ is typically a logical predicate stating the `rdf:type` of the elements of the set they describe, for example $\mathcal{R}(x) \leftrightarrow$ (x `rdf:type someClass`) or restricting the elements of the set to those with a certain property, e.g. $\mathcal{R}(x) \leftrightarrow (\exists y :$ x `someProperty` $y)$. Each $s \in S$ resp. $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$ resp. $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. A complex similarity measure σ can be obtained by combining $\sigma_1, ..., \sigma_n$ with certain operators, in this work restricted to `AND`, `OR` and `MINUS` [24]
The height h of a link specification is defined as follows: If the link specification is an atomic similarity measure $h = 0$. If two complex metrics σ_1, σ_2 are combined with an operator $h = max(|\sigma_1|, |\sigma_2|) + 1$, with $|\sigma|$ being the height of σ.

Definition 3 (Link Discovery as Classification) *"Given the set $S \times T$ of possible matches, the goal of link discovery is to find a classifier $\mathcal{C} : S \times T \to \{-1, +1\}$ such that $\mathcal{C}$ maps non-matches to the class -1 and matches to $+1$" [24, p.4].*

For this work we assume that classifiers operate in an n-dimensional similarity space $\mathfrak{S}$, with n being the number of similarity metrics $\sigma_i(s,t)$ where $s \in S, t \in T$. Each classifier can be modeled through a specific function $\mathcal{F}$, that returns +1 **iff** it holds and -1 else. The function $\mathcal{F}$ is constructed in the same manner as a link specification in 2. Therefore the classification function is

$$\mathcal{F}(s,t) = \begin{cases} +1 & \text{if } \sigma(s,t) >= \theta \\ -1 & \text{else} \end{cases} \tag{2.1}$$

where $\sigma(s,t)$ is the complex similarity metric with the threshold θ and $s, t \in S \times T$.

2.3.2 Frameworks

Several frameworks have been devised to tackle this problem. KnoFuss [28] is a knowledge fusion architecture that is based on research on problem-solving methods. The fusion process has methods for subtasks that can be

combined. Depending on the domain and task the best method is selected. SILK [38] provides a declarative language for specifiying link conditions and due to the fact, that it accesses data sources through the SPARQL [9] protocol it can be employed in distributed environments without the need to replicate datasets locally. Furthermore, it increases performance and reduces network load by implementing a number of various caching, indexing and entity pre-selection methods. The distributed instance matching system Zhishi.links [29] applies a distributed framework to process semantic resources and index them. For calculating similarities between two resources it utilizes scalable matching strategies. Our approach is embedded in the LIMES (Link Discovery for Metric Spaces) [23] framework. It is a lossless and time-efficient approach exploiting the triangle inequality of metric spaces to ignore large numbers of instance pairs, that cannot satisfy a given link specification.
An in-depth comparison and overview of current link discovery frameworks can be found in [22].

2.3.3 Machine Learning

In the following, we will give a brief overview of some machine learning approaches for the link discovery problem. Machine learning is the study of computer programs that automatically improve their performance using experience [21]. Due to the restricted nature of this work we will limit ourselves to the part of this field that is relevant to our approach. A general overview an be found in [21, 17]. As we have seen in Definition 3 link discovery is a binary classification problem and therefore can be tackled with long-standing machine learning approaches such as support vector machines [37, 36], artificial neural networks [10], genetic programming [15] etc.
Most approaches use supervised techniques, this means a labeled user input is necessary. There are mainly two different kinds of supervised learning approaches: *active* and *batch*. The first is an interactive process where the learning algorithm selects a number of instances that are presented to an oracle (e.g. a human user) for labeling, and this way increases its training data size iteratively. The latter takes an initial amount of instances which are the only input used throughout the whole learning process.
There are several implementations of machine learning algorithms for link discovery. EAGLE [25] utilizes genetic programming, EUCLID [26] is an unsupervised learning algorithm, WOMBAT [35] uses an upward refinement operator to traverse the space of link specifications. There have also been efforts to combine unsupervised learning techniques and genetic algorithms such as [27].
Ensuing, we will explicate how decision tree learning can be used for our problem.

[9]a RDF query language

Chapter 3

Decision Trees

Decision trees are a commonly used method for data mining and machine learning. The learning of a tree is done by a process called *recursive partitioning*, where a source set is split into subsets based on the value of an impurity function. We will first have a look at the general concept, goals and implementations, and then consider each step in the learning process in detail.

3.1 General Concept

Decision tree learning is a form of *supervised learning*. Each instance in the training data is labeled with a class. We use this data to find patterns, that help us put unseen (therefore also unlabeled data) into the right class. We will restrict our analysis in this work to *classification trees*, which means we have a number of classes we want to sort data into. Apart from classification trees there are also *regression trees*, which try the same with continous values instead of discrete classes.[10]
The first step of building a tree is dividing the training data into 2 or more sub-samples that do not overlap. This is done by choosing an attribute of the data as *splitting attribute*. Each instance in the root of the tree, i.e. the original data, is sent into a node depending on its splitting attribute value. The most important part of this process is the choice of the splitting attribute, where the goal is to pick an attribute, that will partition the training data into subsets where all the class variables have the same value. The value of the splitting attribute on which the data gets divided follows the same idea. This process is repeated recursively on each node until the stop condition is reached: all nodes have been either split or are leaves [40, pp.99ff.].

3.1.1 Goals

Limiting the size of the tree generated is one of the core goals with the leaves classifying instances most accurately (this is called *leaf purity*). The reason for preferring smaller trees is that on the one hand smaller trees are easier to interpret and on the other hand the leaves hold more information, since they hold more instances. Building a tree that is both small and has a high leaf purity is generally a difficult task, since these two goals often contradict each other and finding an optimal tree is often impossible given the vast solution space [32, p.20]. As shown in [13] constructing optimal binary decision trees is even *NP-complete*.

[10]A notable implementation of this can be found in **CART** [6]

3.1.2 Implementations

Naturally over the years there have been several approaches tackling this problem. A pioneering work in decision tree learning has been the *ID3* algorithm from Ross Quinlan [31], which he later modified and improved to create the popular *C4.5* algorithm [32]. Another notable implementation has been done by Leo Breiman, Jerome Friedman, Richard Olsen and Charles Stone with *CART*, which stands for **C**lassification **A**nd **R**egression **T**rees [6]. Decision trees have also been used in connection with other machine learning techniques, for example to initialize a neural network [41], with genetic algorithms [30] and simulated annealing [7].
In the following we will take a look at all the steps that have to be taken to result in a good tree. We will mainly focus on the method used by *C4.5* and its successors.

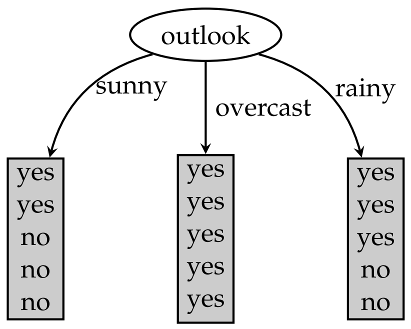
3.2 Splitting a Node

The goal of a node split is to increase the purity of the subsets compared to the purity of the original set. The ideal case for this is if the subsets all have the same value for the class attribute, making them completely pure leaves. Each attribute of the data set is a split attribute candidate. There is a difference however between numerical and nominal attributes. The splits on nominal attributes are n-ary and are of the form: "is $x = c_i$?" with $c_i \in \{c_1, ..., c_n\}$ being a value of the n possible values for this attribute. Numerical attributes produce binary splits since the test is: is "$x <= c$?" with c being the value on which the split attribute will be split [40, p.99,193]. Let's have a look at a classic example to clarify.[11] In Table 3.1 you can find data which is used to determine if one should or shouldn't play tennis (this is the class variable), depending on the weather conditions.

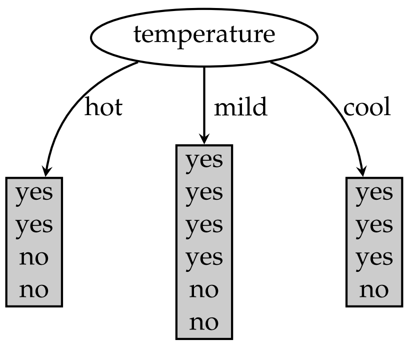
Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rainy	71	91	TRUE	no

Table 3.1: *Weather data*

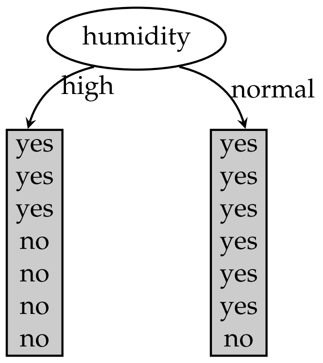
[11]This example is taken from [32]



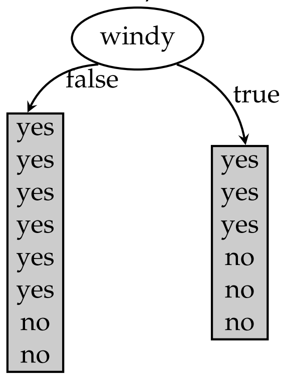
***(a):** outlook*



***(b):** temperature*



***(c):** humidity*



***(d):** windy*

***Figure 3.1:** tree stumps for weather data*

Since we have four attributes we have four choices for the first split. The result can be seen in Figure 3.1. Picking the best split means choosing the split that produces the purest child nodes. There are different measures for purity. We will have a look at the ones used in C4.5: *information gain* and *gain ratio*. Other notable purity measures are the *gini index* which is used in CART [6] and chi-square used by Hart [9] and Mingers [19] [20]. An overview and comparison of measures can be found in [18].

3.3 Measures of Purity

3.3.1 Information Gain

This measure is based on the change of *entropy* after splitting, which is measured in *bits*. Entropy measures the average amount of information that is needed to decide if a new instance belongs to a class c. So before looking at the formula let us point out the fact, that this measure should be *zero* if all the instances in a node belong to the same class, and reach a *maximum* if the instances are equally divided over the classes. The entropy function satisfies these properties:

$$entropy(p_1, p_2, ..., p_n) = \sum_{i=0}^{n} -p_i \log p_i \tag{3.1}$$

Since the arguments $p_1, ..., p_n$ are fractions that add up to one, the minus sign in the formula is needed to make sure the resulting entropy is positive. If we take for example the left leaf of the outlook node Figure 3.1a the information is:

$$info([2,3]) = entropy(2/5, 3/5) = -2/5 \times \log 2/5 - 3/5 \times \log 3/5 = 0.971 \text{ bits}$$

The parameter is the number of instances for each class, which is two for *yes* and three for *no*. For the other leaves of this node we get:

$$info([4,0]) = 0.0 \text{ bits}$$
$$info([3,2]) = 0.971 \text{ bits}$$

To calculate the average information value of these we have to take into account how many instances go down each branch.

$$info([2,3],[4,0],[3,2]) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 = 0.693 \text{ bits}$$

The result tells us the amount of information that is necessary to determine the class of a new instance if we have the tree structure from Figure 3.1a. To calculate the *information gain* we have to determine the information of the root node T and subtract the information of the child nodes $T_1, ..., T_n$. The formula we used to calculate $info([2,3],[4,0],[3,2])$ is

$$info_x(T) = \sum_{i=0}^{n} \frac{|T_i|}{|T|} \times info(T_i) \qquad (3.2)$$

where $|T|$ denotes the number of instances in this node. So if we split the node T on attribute x our information gain is

$$gain(x) = info(T) - info_x(T) \qquad (3.3)$$

For our outlook node the root had nine *yes* and five *no* instances, which leads us to the information value of

$$info([9,5]) = 0.94 \text{ bits}$$

So if we split on outlook we get an information gain of

$$gain(\text{outlook}) = info([9,5]) - info([2,3],[4,0],[3,2]) = 0.94 - 0.693 = 0.247 \text{ bits}$$

If we calculate the information gain for the other attributes as well, we can see that outlook is actually most promising, since it has the highest information gain:

$$gain(\text{temperature}) = 0.029 \text{ bits}$$
$$gain(\text{humidity}) = 0.152 \text{ bits}$$
$$gain(\text{windy}) = 0.048 \text{ bits}$$

The considerations we made here can also be found a bit more detailed in [32, pp.20-22] and [40, pp.103ff]. Although this procedure gives quite good results, it has a strong bias towards branches with a huge number of child nodes. To counter this a different measure is used.

3.3.2 Gain Ratio

Consider for example an attribute *ID* which gives every instance a unique identification. If we split on this attribute every node will only contain one case, so the nodes are necessarily pure and $info_x(T) = 0$. Splitting on this attribute therefore would lead to a maximum information gain, so we would always split on this. The resulting decision tree however would be pretty useless for predicting unknown instances and would give no information about the structure of the decision [40, p.104]. To counter this effect a kind of normalization is used called *gain ratio*.

$$gain\ ratio(x) = gain(x)/split\ info(x) \tag{3.4}$$

where $split\ info(x)$ is defined as

$$split\ info(x) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log\left(\frac{|T_i|}{|T|}\right) \tag{3.5}$$

In contrast to the *information gain*, which measures information that is needed for classification in that node, $split\ info$ represents the potential information that is generated by dividing a node into n child nodes. So in $gain\ ratio$ we have the information that is useful for classification, generated by the split [32, p.23]. If we want to calculate the gain ratio for our outlook split, we first have to calculate the split info:

$$-5/14 \times \log(5/14) - 4/14 \times \log(4/14) - 5/14 \times \log(5/14) = 1.577\text{bits}$$

As we have already calculated before the information gain for outlook is 0.246, so the gain ratio is 0.246 / 1.577 = 0.156.
The *gain ratio* seems to be a good fix for our *ID* problem, however in some cases it overcompensates and an attribute is chosen solely on the fact, that it has a small value of intrinsic information. A good workaround for this is to choose attributes that have at least an average information gain and maximize gain ratio [40, pp.105ff].

3.4 Missing Values

Most real datasets have missing values. This can occur for a number of reasons: malfunctioning equipment, participants in a study, that refuse to answer a question etc. Most machine learning algorithms treat missing values with the implicit assumption, that these don't have any particular significance. But there can be systematic reasons for the absence of these values, that cannot be taken into account this way. In this case it might be fruitful to see *missing* as another possible value [40, p.58]. Another simple approach would be to use the most popular branch of the tree if a value is missing in an instance.
A more refined approach is to fictitiously split an instance and send a part of it down each branch until we reach the leaves. We use a numeric weight between zero and one for this split, depending on the number of instances going down that branch. All weights however must sum up to one. This can be repeated at lower nodes. Finally, at the leaves, the different parts of an instance have to be recombined using the weights. The information gain and gain ratio can also work with partial instances [40, pp.194f].

3.5 Pruning

Usually a tree contains unnecessary parts and may overfit the data. A technique to counter this and simplify the tree before using it is called *pruning*. There are two different types of pruning we are going to look at: *pre-pruning* (or *forward pruning*) and *post-pruning* (or *backward pruning*).

3.5.1 Pre-Pruning

This approach basically involves having a higher restriction for stopping rules. Whereas normally construction of the tree is stopped, if all the leaves are pure, or all the feature values are the same, in this case we terminate sooner. In pre-pruning you stop if the number of instances falls below a predefined threshold or if expanding the current node does not improve impurity. While it is seems very attractive not to develop subtrees that are going to get pruned afterwards, it is very difficult to find good stopping rules: "Depending on the thresholding, the splitting was either stopped to soon at some terminal nodes or continued too far in other parts of the tree" [6, p.37]. This is why most decision tree builders prefer post-pruning, although if runtime is of concern pre-pruning can be a feasible alternative [40, p.195]

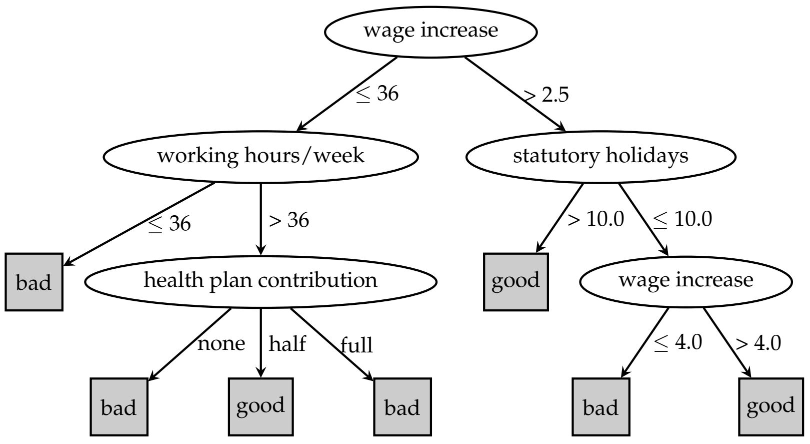
3.5.2 Post-Pruning

In contrast post-pruning takes place after the tree is fully developed. There are two different approaches for this type of pruning: *subtree replacement* and *subtree raising*.
We start with the more popular subtree replacement. The idea is to replace a subtree with a leaf. Naturally this will decrease the accuracy of this tree on the training set, but it might be beneficial for other data. In practice we work from the leaves upwards to the root and on each node decide if we should replace it with a leaf. If we have a look at Figure 3.2 we see how subtree replacement works on example data concerning labour negotiations.[12] The node 'working hours/week' in Figure 3.2a is pruned, by replacing it with the leaf 'bad' in Figure 3.2b.
Subtree raising is more complex. Instead of replacing a subtree with one of its leaves, we replace a subtree with one of its subtrees, therefore "raising" the latter. This is a potentially time-expensive operation and is generally reserved for the most popular branch [40, p.196]. If we look in Figure 3.3 we can see how the node 'B' in Figure 3.3a is replaced with 'C' in Figure 3.3b. Note, that the instances in the leaves have to be reclassified, so the leaves of 'C' are marked with primes to highlight, that they are not the original ones [40, pp.195f].

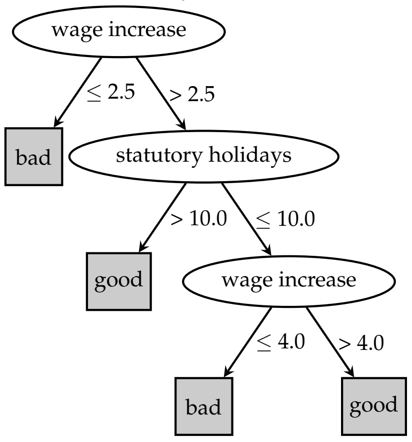
3.6 Error Rates

So how do we make the decision to either replace an internal node with a leaf, or one of its subnodes? To achieve the best results we need a way to estimate the error rate. Since the tree is fit precisely to the training data,

[12]This example is taken from [32]



***(a):** unpruned tree*



***(b):** pruned tree*

***Figure 3.2:** subtree replacement*

it is unreasonable to use the training set error as error estimate. A way to avoid this problem is called *reduced-error pruning* and it consists of holding back part of the training set to estimate the error rate. The obvious downside of this is that we have less training data while building the tree. C4.5 uses a different approach, making an error estimate on the training data itself. While Quinlan concedes, that his heuristic has "questionable underpinning", he also adds that the "estimates that it produces seem frequently to yield acceptable results" [32, p.41]. The basic idea is to choose the majority class for all instances that reach a node and get a number of errors E (misclassified instances) out of the total number of instances N. The next step is assuming that the actual probability of errors at a node is q and a Bernoulli process[13] with parameter q is responsible for the N instances, of which E instances are errors. What we do with this is a pessimistic error estimate using the upper confidence limit. This goes as follows.[14]

[13]"In statistics, a succession of independent events that either succeed or fail is called a *Bernoulli process* " [40, p-150]

[14]Due to the restricted nature of this work we can only give a brief summary of the process and cannot give an in-depth analysis of the statistical foundations of these calculations

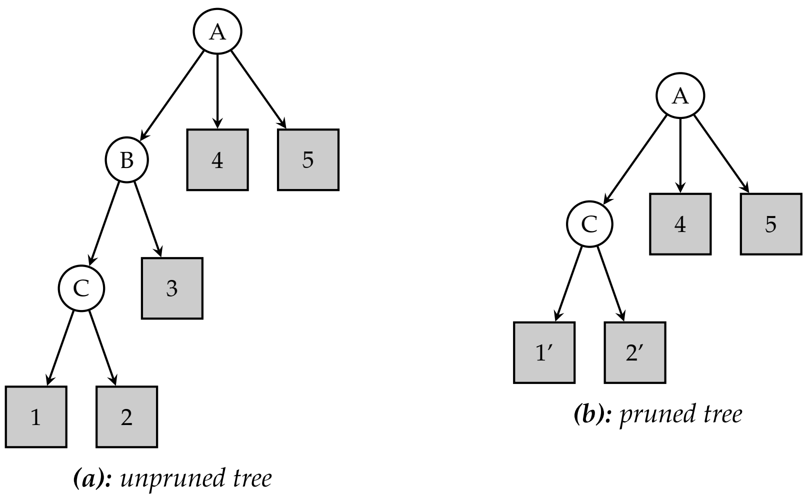


Figure 3.3: *subtree raising*

Let c be a confidence for which we find a confidence limit z using

$$Pr\left[\frac{f-q}{\sqrt{q(1-q)/N}} > z\right] = c \tag{3.6}$$

where N is the number of instances, f the observed error rate, which is calculated as *E/N*, and q the true error rate. With the upper confidence limit for q we can calculate a pessimistic error estimate for the error rate e at the node

$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \tag{3.7}$$

Since we want only the upper confidence limit a '+' is used before the square root, instead of a '±' which normally is in this place. If we want to find out if we should prune a node, we calculate the error estimates of all leaves. The result is then combined taking into account the number of instances in each leaf and using this as weights. If the error estimate of the node is less than the combined error estimate of the children it is pruned [40, pp.197f].

To see how all this is applied we will look at our labor negotiations example. In Figure 3.4 we see a part of our unpruned tree from Figure 3.2a with the number of instances added to each leaf. The default confidence level c is set to 0.25 in C4.5, so our z is 0.69. On the lower left leaf we have E=2, N=6, which gives us f=0.33. If we put this into Equation 3.7 we get e=0.47. So instead of using the training set error rate of 33% we use the pessimistic estimate of 0.47%. Considering we have two classes (*bad* and *good*) this is really pessimistic, since it would be terrible to exceed an error rate of 50%. But if we take a look at the next leaf to right, it is even worse. We have E=1 and N=2 so e=0.72. The third leaf has the same value for e as the first. Since we have calculated all the leaves we now have to determine the combined error estimate and take into account the number of instances they cover (6:2:6), which gives us 0.51. The next step is calculating the error estimate for the parent node 'health plan contribution'. It covers five bad and nine good examples, so f=5/14. Plugged into our formula we get e=0.46. Since

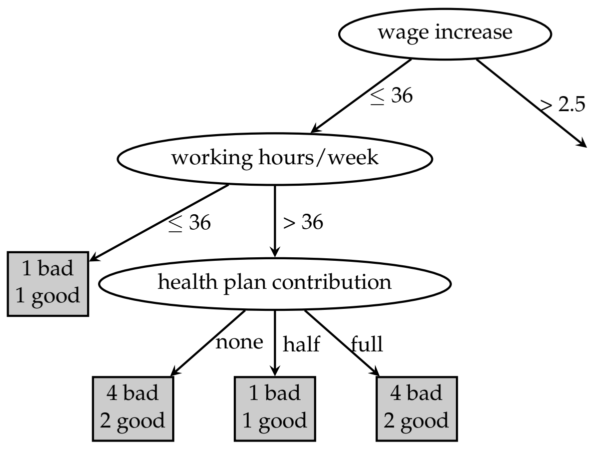


Figure 3.4: *part of labor negotiation tree*

this is less than the combined error of the leaves they get pruned. We go the next node 'working hours/week' and start with the left node. It has E=1, N=2 and therefore e=0.72. For the right child we just calculated that e=0.46. Combining them with the ratio 2:14 it gives us an error rate higher than the error rate of the parent node so it will be pruned as well.

3.7 Further Refinements

The last non-commercial release of C4.5 was Release 8 and it incorporated an MDL-based[15] adjustment to information gain, if it's used on numeric attributes. This means if there are S candidate splits, $log_2(S)/N$ is subtracted from information gain, with N being the number of instances at the node. After subtraction the information gain may be negative. Since the tree stops growing if there are no attributes with positive information gain this is not only designed to prevent overfitting, but is also a form of pre-prunning [40, p.201].

[15]"minimum description length principle" introduced in [33]

Chapter 4

Approach

We will now have a look at how we can use J4.8, a Java reimplementation of C4.5 [40, p.454], to learn link specifications. We start with an overview and then delve into each step.

4.1 Overview

Algorithm 1: Overview

```
Require: - Specification of two knowledge bases KS and KT
         - Mapping trainingMapping or LinkSpecification
           defaultLS
Output: LinkSpecification learnedLS
Get set $S$ and set $T$ of instances as specified in $KS$ respectively $KT$
repeat
    if learnedLS is null then
        if trainingMapping == null then
            trainingMapping ← $execute$(defaultLS)
        end
    end
    else
        Mapping $m$ = $execute$(learnedLS $-\delta$)
        trainingMapping ← user labels $n$ link candidates $(s, t) \in m$
    end
    if previouslyPresentedCandidates.size > 0 then
        trainingMapping.$union$(previouslyPresentedCandidates)
    end
    previouslyPresentedCandidates.$add$(trainingMapping)
    trainingData ← $createTrainingInstances$(trainingMapping)
    learnedLS ← $learn$(trainingData)
    $checkIfThereWasABetterLSBefore$(learnedLS)
until user termination;
```

Algorithm 1 gives an overview of the approach implemented. In the first iteration, *learnedLS* is null. If a link specification was provided instead of a *trainingMapping* , we execute it and use the result as *trainingMapping*. All instance pairs with mapping values higher than a threshold κ get labeled as +1. In *previouslyPresentedCandidates*, which is *null* in the first iteration, we save the instances labeled by the user. The *createTrainingInstances* function parses the mapping into *ARFF* format, which is used by weka [12], the framework where J4.8 is implemented. The result is our *trainingData*.

This data is taken as input for the *learn* function, which builds a decision tree. This tree is parsed into the *learnedLS*. If the user does not stop the loop, another iteration begins. Since *learnedLS* is not *null* this time, we subtract δ from all thresholds in *learnedLS* and execute the resulting link specification to obtain the mapping m. We take the n most informative link candidates from m for user labeling and the labeled candidates become our new *trainingMapping*. This time the size of *previouslyPresentedCandidates* is bigger than 0, so we add these instances to our *trainingMapping* by using a union operation. The remaining part of the algorithm is identical to the first iteration. Additionally to the described *active learning* variant, where a user is repeatedly asked to label data, there is also a *batch learning* variant, where only the initial training mapping is used.
We will now have a closer look at all the functions, that are used in the algorithm. To exemplify what the classifier does, we will use two knowledge bases named *person11* and *person12* from the OAEI 2010 benchmark[16]. The initial *trainingMapping* consists of 10 positive and 10 negative examples.

4.2 Creating Training Instances

We will start with the *createTrainingInstances* function. Our input is a mapping, consisting of instance pairs, that are labeled by the user as +1 or -1. The output will be a set of instances, which will be referred to as *training Instances*. An instance has two kinds of attributes: the class attribute (positive if labeled by the user as +1, negative otherwise) and what we will call a *measure attribute*. Each instance has one class attribute and $M \times PP$ *measure attributes*, M being the measures that are permitted for a property pair PP. A measure is used on the property pair and the result is the value of the corresponding measure attribute. For each element in the *trainingMapping* we create a training instance. This means we have used all similarity measures on all properties of a mapping pair and this was done for all mapping pairs.
Applied to our persons example the training instances object looks like this:[17]

```
@relation link

@attribute cosine(x.surname,y.surname) numeric
@attribute jaccard(x.surname,y.surname) numeric
@attribute jaro(x.surname,y.surname) numeric
[...]
@attribute cosine(x.has_address,y.has_address) numeric
@attribute jaccard(x.has_address,y.has_address) numeric
[...]
@attribute cosine(x.given_name,y.given_name) numeric
@attribute jaccard(x.given_name,y.given_name) numeric
[...]
@attribute match {positive,negative}
```

[16]`http://oaei.ontologymatching.org/2010/im/` OWL Data Track - PR: Person1
[17]For better legibility `http://www.okkam.org/ontology_person1.owl#` has been shortened to `x`, resp. `http://www.okkam.org/ontology_person2.owl#` to `y`

```
@data
0,0,0.944444,0.166667,0,0.666667,0.5,[...]positive
0,0,0,0,0,0.666667,0.5,0.93088,0.734694,[...]negative
[...]
```

The output we can see here is in *ARFF* format [40, pp.52-56]. First listed is the name of the relation followed by all the attributes. Each line below "`@data`" represents one instance with its commaseperated values. In our example, the first pair from the *trainingMapping* is `http://www.okkam.org/oaie/person1-Person740` and `http://www.okkam.org/oaie/person2-Person741` and the cosine similarity between the values of the corresponding surname properties "begic" and "begdic" is 0, which you can see in the first value of the first line of the instance. Jaccard similarity between the surnames is also 0, for Jaro it's 0.944444 and so on. The last attribute is the class attribute, so we can see this pair was labeled as +1.

4.3 Learning

These instances are used to produce a decision tree as was explained in chapter 3. There are a number of options that can be provided by the user to tweak the learning behaviour:

- **training data size**: the number of examples that have to be labeled by the user during active learning
- **use unpruned tree**: disable pruning
- **collapse tree**: handles pruning based on classification error on the training data (should be turned off for an unpruned tree)
- **pruning confidence**: the confidence level used for the pruning decision
- **reduced error pruning**: enables reduced error pruning (cannot be used combined with pruning confidence)
- **fold number**: number of folds used for reduced error pruning
- **subtree raising**: perform subtree raising
- **clean up**: clean up after building the tree (saves memory)
- **laplace smoothing**: use laplace smoothing[18] for predicted probabilities
- **mdl correction**: use mdl correction for predicted probabilities
- **seed**: seed used for randomizing the data in case reduced-error pruning is used
- **max linkspec height**: maximum height of the link specification

[18]This is a technique to ensure that attributes that occur zero times get a probability that is a bit higher than zero. This is done to prevent a single zero value turning a whole equation to zero. Most of the times 1 is added to all counts [40, p.93]

4.4 Parsing the Decision Tree into a Metric

4.4.1 General procedure

A decision tree can be represented as logical formula in disjunctive normal form [34]. We use this knowledge to build a metric, by using the AND, OR, and MINUS operators. The parser starts at the root of the tree and tries to find positive leaves via depth first search. All the nodes we have passed on our way are combined as measures with an AND. We take the threshold of the edge leading to the positive leaf as the threshold τ for our atomic measure. We have a lower limit λ for any τ. In this work we set λ to 0.1. The reason for this is, because below a certain threshold measures can take a long time to execute and the results obtained don't justify the time invested. On our way traversing the tree towards positive leaves, we might come across an edge labeled with "$<= \tau$". To represent this in our link specification, we somehow must *negate* the measure from the node we came from. This is done by taking the measure and connecting it with itself via a MINUS operator, once with τ as threshold and the other with λ. After we have done this for all positive leaves each metric produced is combined via the OR operator.
It is important to note, that there are a few optimizations done while parsing. On our way to positive leaves we often take paths multiple times, so we can use this in combination with the distributive law to make trees smaller. We will see this in an example later. We also ignore irrelevant subtrees: If we pass an edge labeled with "$<= 0$" the subtree is replaced with a negative leaf. The reasoning is the same as setting a lower limit for τ. It is also possible to provide a maximum height h for the link specification. If the link specification is bigger than h the parser replaces the internal nodes at h with their non-negated leaves that are near h. We refer to this as *cutting the link specification*.
Formally the procedure can be seen in algorithm 2. The function *nodeToMeasure* parses an atomic tree, taking the threshold of the edge leading to the positive leaf as τ for the metric, *addRoot* adds the root node of t with AND to the metric given as second argument, *addOR* combines metrics with the OR operator.
For a better understanding we will have a look at some trees and how they are parsed to metrics.

4.4.2 Examples

The learned tree for the persons example can bee seen in Figure 4.1. This is a trivial case, since we only have one node which becomes our measure. Our metric therefore is $\texttt{cosine}((x.given\_name, y.given\_name), 0.1)$.

We can see a more complex example in Figure 4.2 . The parser starts on the node *qgrams(name,name)*. Because the left child is a negative leaf we are only interested in the right and obtain $\texttt{qgrams}((x.name, y.name), 0.1)$. On the next node, we find a positive leaf on the right and. We turn the node into a metric and add it to our previous node using AND which **would** conclude in $\texttt{AND}(\texttt{qgrams}((x.name, y.name), 0.1), \texttt{trigrams}((x.name, y.name)0.2))$. But if we go further, we see that our current node has a subtree with a threshold that is *not* $<= 0$ and leads to a positive leaf. So we first add

Algorithm 2: parseTree

```
Require: decision tree t
Output: LinkSpecification ls
if isAtomic(t) then
    return nodeToMeasure(t)
end
if t.leftThreshold > 0 then
    leftMetric = parseTree(t.getLeftSubtree())
end
rightMetric = parseTree(t.getRightSubtree())
leftMetric = addRoot(t, leftMetric)
rightMetric = addRoot(t, rightMetric)
if leftMetric != null && rightMetric != null then
    ls = addOR(leftMetric, rightMetric)
end
else if leftMetric != null then
    ls = leftMetric
end
else if rightMetric != null then
    ls = rightMetric
end
else
    return null
end
return ls
```

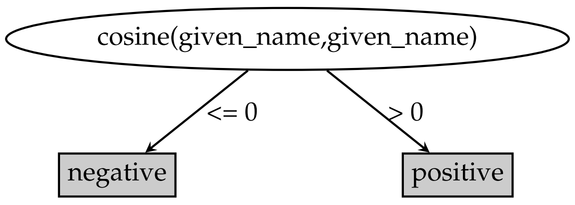


Figure 4.1: *Simple tree*

this subtree to `trigrams`$((x.name, y.name), 0.2)$ with an OR operator and we also have to add the negation of `trigrams`$((x.name, y.name), 0.2)$ to `jaccard`$((x.age, y.age), 0.2)$ using AND, since we passed the edge labeled with "<= 0.2 to get to `jaccard`$((x.age, y.age), 0.2)$. **Now** we can add OR (`trigrams`(...), AND(`jaccard`(...), MINUS(`trigrams`(...)`trigrams`(...)))) to our first measure. Since there are no leaves left the parser terminates. The result can be seen in Figure 4.3.

4.5 Check if there was a better link specification before

Because we want to give back the best link specification we learned, we have to keep track of what we have already learned and how good it is. This is implemented in the *checkIfThereWasABetterLSBefore* function seen in algorithm 1. If we have a new link specification, we first check if we

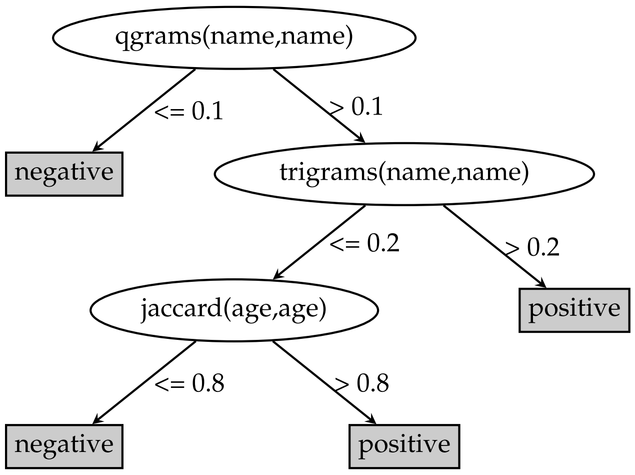


Figure 4.2: *Complex tree*

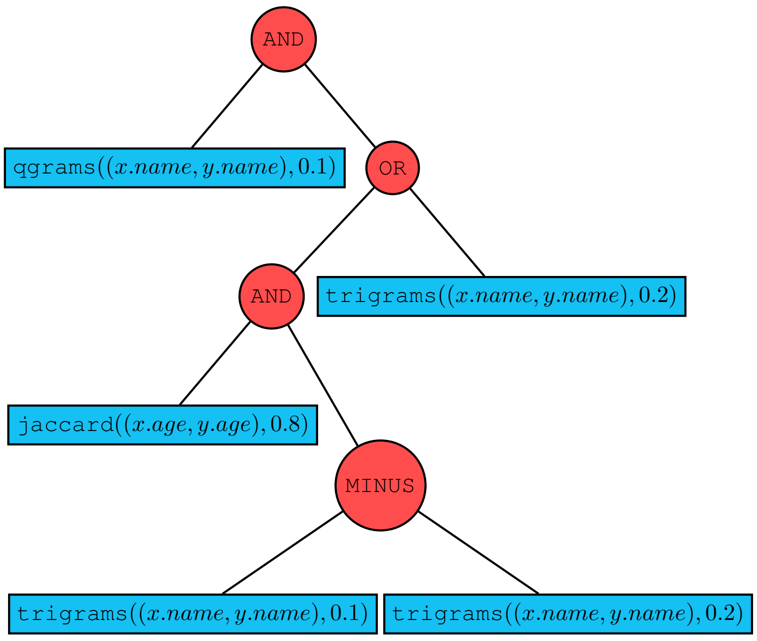

AND
qgrams((x.name, y.name), 0.1)
OR
AND
trigrams((x.name, y.name), 0.2)
jaccard((x.age, y.age), 0.8)
MINUS
trigrams((x.name, y.name), 0.1)
trigrams((x.name, y.name), 0.2)

Figure 4.3: *Complex metric*

have already seen it. If this is the case we increase its threshold by γ until we get a link specification we have not seen yet. Setting $\gamma = 0.05$ has proved to be reasonable. Before this was added, most of the times only the threshold changed during active learning iterations. Because we want to have good results with as little iterations as possible, manually raising the threshold can be considered a shortcut. We calculate the *PseudoFMeasure* of the mapping we get from executing the learned link specification and save the results. For the batch learning variant we check all link specifications with thresholds bigger or equal to our originally learned link specification, because we only have one iteration.

4.6 Most Informative Link Candidates

Since we want the labeling to be as effective as possible, we present the user those link candidates that the algorithm is most unsure about. For this, we

use a strategy similar to *RAVEN* [24].
As we know from 2, every link specification has a (complex) similarity measure σ with a threshold θ. If we execute a link specification, we check for every pair $(s, t) \in S \times T$ if the value of $\sigma(s, t)$ is bigger or equal to θ. By doing this, we draw a boundary in the similarity space $\mathfrak{S}$ with the pairs that satisfy $\sigma(s, t) >= \theta$ considered +1 by our classifier, all others -1. The pairs that are closest to this boundary are the ones the algorithm is most unsure about and therefore the most informative if labeled.
Practically this is done as follows:

1. take our original link specification *ls*, subtract δ from all thresholds and get a new link specification *deltaLS*
2. obtain *delta mapping* by executing *deltaLS*
3. calculate a compound measure ϕ for each pair in *delta mapping*
4. return n link candidates, that have not previously been shown to the user and have the smallest ϕ

We have to subtract δ from *ls* because otherwise our mapping would not contain pairs that are barely classified as *non-matches*, i.e. approaching the boundary from below. Bear in mind, that for measures connected with `MINUS` we have to **add** δ to the threshold that is not λ, since this complex metric represents a negation of that measure. The compound measure used can be seen in Equation 4.1, with n being the number of measures used in *deltaLS*, $\sigma_1, ..., \sigma_n$ the atomic measures in *deltaLS* and $\tau_1, ..., \tau_n$ the corresponding thresholds in *ls*.

$$\sum_{i=1}^{n} |\sigma_i(s, t) - \tau_i|^2 \tag{4.1}$$

Chapter 5

Experiments and Results

In this chapter we will examine how well our approach is able to find good link specifications. First we will explain our experimental setup and then we will look at the two experiments we have conducted. The first experiment tests our algorithm in the active and batch learning variant against three other classifiers, the second inspects the effect user feedback has in active learning iterations.

5.1 Experimental Setup

All experiments were carried out on nine benchmark datasets. Three were synthetic datasets from OAEI 2010 benchmark[19] and four were real-world datasets from [14]. The two remaining datasets were *Drugs* and *Movies* taken from [25]. All experiments were carried out on a 64-core 2.3 GHz Server running Oracle Java 1.8.0_77 on Ubuntu 14.04.4 LTS, with each experiment assigned 20 GB of RAM.

5.1.1 Measures

The measures we used were: *Precision*, *Recall* and *F-measure*. Precision is, in our case, the number of correctly classified links (true positives) divided by the number of all links (true positives and false positives) in a mapping. Recall on the other hand is the number of true positives divided by the number of all links between the knowledgebases (true positives plus false negatives). F-measure describes the harmonic mean between precision and recall: $\frac{2 \times P \times R}{P + R}$. Apart from the quality of the learned link specifications we also measured the time each classifier needs to learn them.

5.1.2 Parameters

The termination criteria for WOMBAT was either finding a link specification with F-measure of 1 or a refinement depth of 10, the coverage threshold was set to 0.6 and the similarity measures used were *jaccard, trigrams, cosine* and *qgrams*. EAGLE's mutation and crossover rates were set to 0.6 and the number of generations were set to 100, which means we used the same parameters as in [25]. EUCLID's parameters were set as presented in [26]: grid size to 5, iterations to 100.
Our decision tree learning was set to our default values:

- collapseTree = true

[19]http://oaei.ontologymatching.org/2010/im/

- pruning confidence = 0.1
- subtree raising = true
- clean up = true
- laplace smoothing = true
- mdl correction = true

For **max linkspec height** we used two different values to determine the effect of this technique, once it was set to 2, the other time it was turned off.

Table 5.1: *Characteristics of the used datasets. S and T stand for Source resp. Target, |perfect result| is the size of the perfect mapping.*

Label	Attributes	$\lvert S \rvert \times \lvert T \rvert$	|perfect result|
Person1	given name surname street number address suburb	2.5×10^5	5.0×10^2
Person2	postcode state date of birth age phone number social security number	2.4×10^5	4.0×10^2
Restaurants	name street city phone restaurant category	7.2×10^4	1.1×10^2
DBLP-ACM	title authors	6.0×10^6	2.2×10^3
DBLP-Scholar	venue year	1.7×10^8	5.3×10^3
Amazon-GP	name description	4.4×10^6	1.3×10^3
Abt-Buy	manufacturer price	1.2×10^6	1.1×10^3
Drugs	name	1.1×10^6	1.0×10^3
Movies	title director	1.1×10^6	1.0×10^3

5.2 Active & Batch Learning Experiment

In the first experiment, we tested our decision tree learning approach against three other classifiers: EAGLE [25], EUCLID [26] (linear version) and WOMBAT [35] (simple version). We used a 10-fold crossvalidation to test our performance. For this the training data is partitioned into ten random subsamples (called *folds*), learning is performed on nine of them and the classifier is then tested on the tenth. For the active learning variant we used a third of the training data for the initial iteration and a labeled feedback of the same size for the next two iterations. The experiment was carried out on 10 times on each dataset and the mean of the values was calculated.

5.3 Active & Batch Learning Experiment Results

The results of this experiment are represented graphically in Figure 5.1, more detailed values concerning F-measure and time can be seen in Table 5.2.1 and Table 5.2.2. We can see that our approach has the best F-measure on six out of nine datasets (outperforming the other classifiers in four cases), and the second best in two of the remaining three datasets, even though on average WOMBAT and EAGLE deliver better results. Concerning time-efficiency, our approach is also faster on six out of nine datasets and only beat by EUCLID in three datasets. It is also evident, that in general cutting of the link specification at height 2 returns the same (or even slightly better) results while saving time: active learning is 44% and batch learning 47% faster if cut. If we take a look at Table 5.2.3 we can see, that our approach has a higher variance than the other classifiers, e.g. on average 97% higher than WOMBAT. A high variance is undesired, because we want to produce stable results.

5.4 User Feedback Experiment

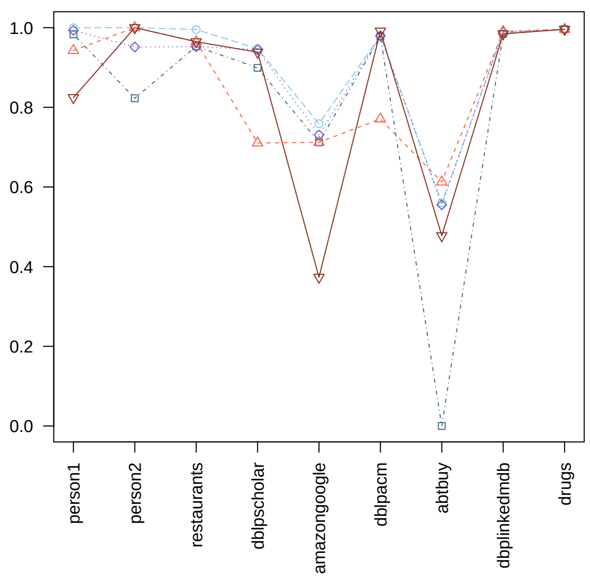
The second experiment was used to determine the effect of user feedback on the active learning variant. We used the same datasets as before, but for each iteration gave the classifier 20 labeled examples. Compared to the first experiment, this makes performance on big datasets harder because the classifier gets a smaller share than before. Parameters were the same as before, **max linkspec height** was set to 2. The experiment was run 5 times and the mean values were taken.

5.5 User Feedback Experiment Results

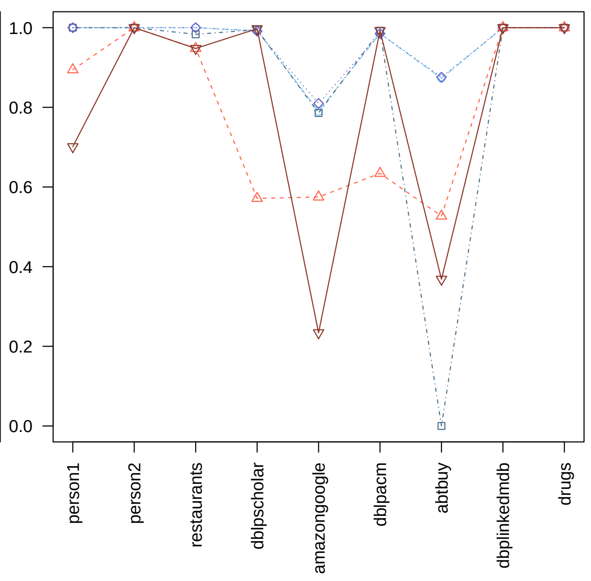
The results are presented in Figure 5.2. We can see a steady increase in F-measure on all datasets until a plateau is reached. Compared to the real-world datasets, the synthetic ones have the steepest rise. In general recall stays the same while precision gets better. In some cases (e.g. abtbuy and amazongoogle) though a decrease in recall can be seen. These are also the datasets with the worst F-measure value. Five out of nine datasets reach F-Scores above 0.8 with 14 iterations. In summary, this means we can get good link specifications with a reasonable amount of feedback iterations (depending on the complexity of the underlying dataset).

5.6 Experiments Summary

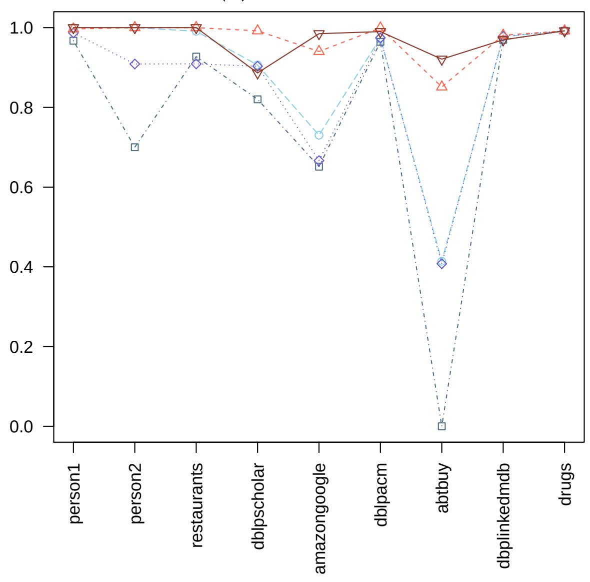
In Table 5.2.1 we can see that the simple datasets using real world data (i.e. drugs and dbplinkedmdb) are no challenge for any of the classifiers we tested, including our decision tree learning approach. This means we are able to compute high quality link specifications for simple datasets. Looking at the synthetic datasets person1, person2 and restaurants we can see that this is a slightly more difficult task, since they contain duplicate records, deliberate differences etc. However, we can also get a F-measure of at least 94% with the active learning variant. Nonetheless, the benefits of decision tree learning can be seen on the real datasets scraped from the web (dblpacm, amazongoogle, abtbuy). While the dblpacm task, which consists of matching two well-structured bibliographic data sources who are at least moderately under human supervision, is of low difficulty amazongoogle and abtbuy are very messy datasets containing "duplicate publications, heterogeneous representations of author lists or venue names, misspellings, and extraction errors" [14, p.487]. Especially on the abtbuy dataset we can see that our approach performs much better than the other classifiers. The various pruning strategies we looked at in section 3.5 give decision trees some resistance to noise in the data, because the impact of outliers usually gets pruned. This can also be seen in the fact, that ActiveC and BatchC perform much better than the uncut versions, because our cutting technique can be seen as a pruning mechanism. The remarks we made in this section can also be observed in Figure 5.2.



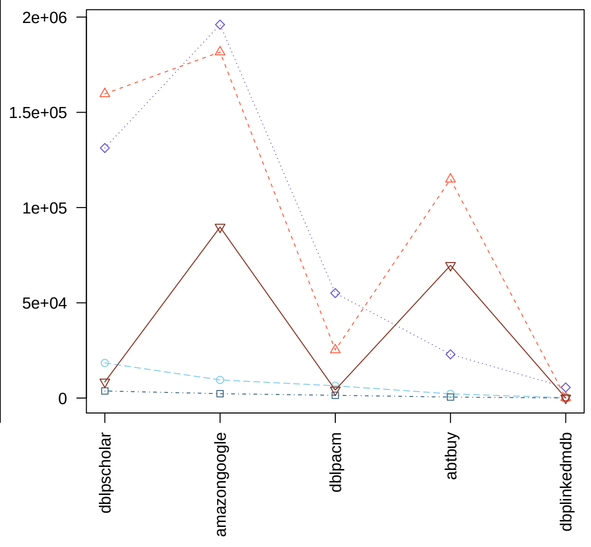
***(a):** F-measure*



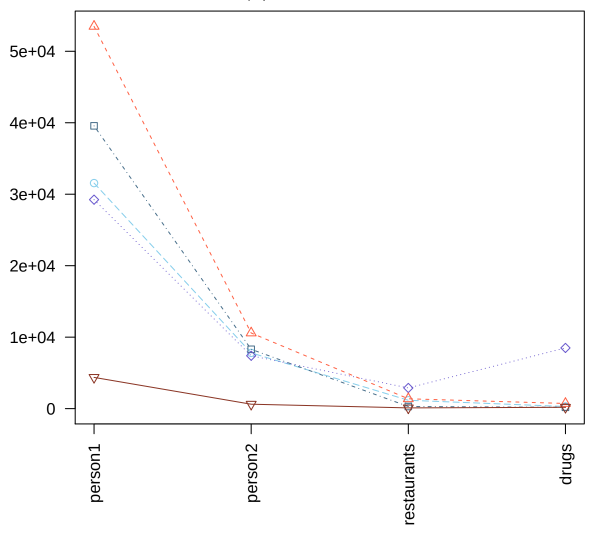
***(b):** Precision*



***(c):** Recall*



***(d):** Time(in milliseconds) for big datasets*



***(e):** Time(in milliseconds) for small datasets*

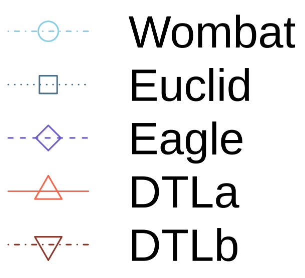
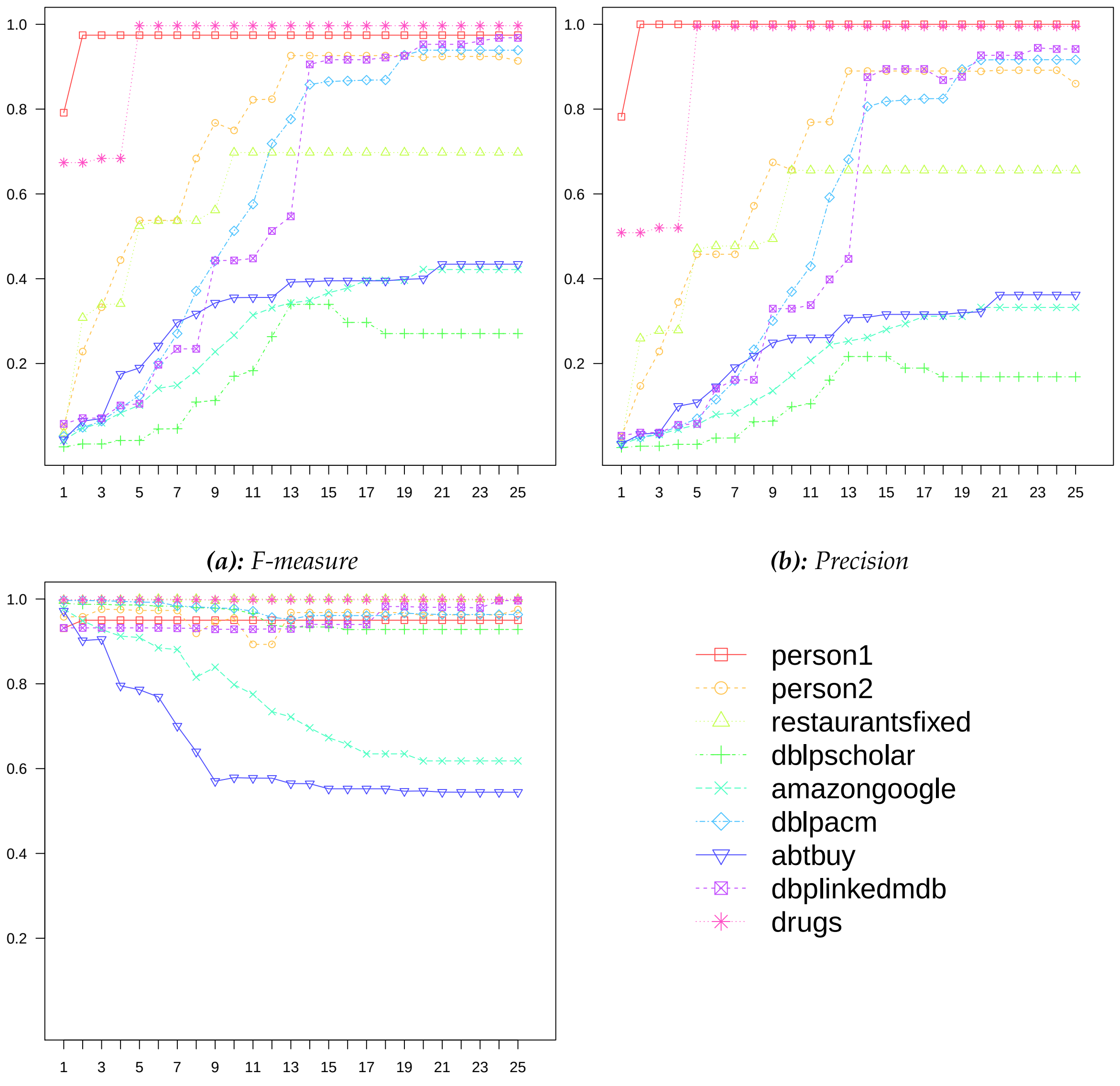


***Figure 5.1:** Results of the active and batch learning experiment. DTLa stands for the active and DTLb for the batch learning variant of decision tree learning*



(a): *F-measure*

(b): *Precision*

(c): *Recall*

Figure 5.2: *Results of the user feedback learning experiment. The x-axis is denoting the corresponding measure and on the y-axis the number of user iterations is displayed*

Table 5.2: *Detailled results of the active & batch learning experiment. Green cells show the best value for each dataset. ActiveC and BatchC represent the cut version of the corresponding learning variant.*

5.2.1: *F-Measure*

Datset	WOMBAT	EUCLID	EAGLE	**Decision Tree Learning**			
				Active	Batch	ActiveC	BatchC
person1	1.0	0.98	0.99	0.94	0.82	0.94	0.82
person2	1.0	0.82	0.95	1.0	1.0	1.0	1.0
restaurants	0.99	0.95	0.95	0.96	0.96	0.96	0.96
dblpscholar	0.94	0.89	0.94	0.71	0.93	0.71	0.93
amazongoogle	0.75	0.71	0.73	0.71	0.37	0.78	0.47
dblpacm	0.97	0.97	0.97	0.77	0.99	0.77	0.99
abtbuy	0.56	0.0	0.55	0.61	0.47	0.73	0.58
dbplinkedmdb	0.98	0.98	0.98	0.99	0.98	0.98	0.98
drugs	0.99	0.99	0.99	0.99	0.99	0.99	0.99
average	0.91	0.81	0.89	0.85	0.83	0.87	0.86

5.2.2: *Time*

Datset	WOMBAT	EUCLID	EAGLE	**Decision Tree Learning**			
				Active	Batch	ActiveC	BatchC
person1	3.2×10^4	4.0×10^4	2.9×10^4	5.3×10^4	4.4×10^3	4.9×10^4	3.8×10^3
person2	7.8×10^3	8.3×10^3	7.4×10^3	1.1×10^4	6.2×10^2	9.2×10^3	6.8×10^2
restaurants	1.2×10^3	3.0×10^2	2.9×10^3	1.4×10^3	1.0×10^2	1.8×10^3	1.4×10^2
dblpscholar	1.8×10^5	3.7×10^4	1.3×10^6	1.6×10^6	8.5×10^4	2.1×10^6	8.9×10^4
amazongoogle	9.5×10^4	2.3×10^4	2.0×10^6	1.8×10^6	9.0×10^5	1.7×10^5	7.3×10^5
dblpacm	6.4×10^4	1.4×10^4	5.5×10^5	2.5×10^5	4.2×10^4	2.1×10^5	4.3×10^4
abtbuy	2.2×10^4	5.2×10^3	2.3×10^5	1.1×10^6	7.0×10^5	7.5×10^4	9.2×10^4
dbplinkedmdb	1.4×10^3	4.1×10^2	5.6×10^4	2.1×10^3	3.3×10^2	1.5×10^3	3.3×10^2
drugs	3.0×10^2	1.9×10^2	8.5×10^3	7.2×10^2	1.9×10^2	5.5×10^2	1.3×10^2
average	4.5×10^4	1.4×10^4	4.6×10^5	5.4×10^5	1.9×10^5	2.9×10^5	1.1×10^5

5.2.3: *Variance of F-Measure*

Datset	WOMBAT	EUCLID	EAGLE	**Decision Tree Learning**			
				Active	Batch	ActiveC	BatchC
person1	0.0	0.0	6.7×10^{-5}	4.7×10^{-5}	0.0	4.7×10^{-5}	0.0
person2	0.0	9.1×10^{-4}	7.5×10^{-4}	0.0	0.0	0.0	0.0
restaurants	2.0×10^{-4}	2.7×10^{-6}	0.0	1.1×10^{-2}	1.1×10^{-2}	1.1×10^{-2}	1.1×10^{-2}
dblpscholar	0.0	4.5×10^{-5}	2.5×10^{-6}	1.6×10^{-2}	1.1×10^{-4}	1.7×10^{-2}	1.1×10^{-4}
amazongoogle	2.1×10^{-5}	0.0	5.2×10^{-5}	1.1×10^{-3}	7.7×10^{-3}	9.2×10^{-4}	1.8×10^{-2}
dblpacm	0.0	2.9×10^{-6}	0.0	5.8×10^{-3}	7.2×10^{-7}	6.5×10^{-3}	7.2×10^{-7}
abtbuy	9.7×10^{-4}	-	6.1×10^{-4}	2.2×10^{-2}	8.3×10^{-3}	2.6×10^{-3}	6.4×10^{-4}
dbplinkedmdb	0.0	0.0	8.5×10^{-6}	1.6×10^{-5}	0.0	1.2×10^{-5}	0.0
drugs	0.0	0.0	0.0	0.0	0.0	0.0	0.0
average	1.3×10^{-4}	1.1×10^{-4}	1.7×10^{-4}	6.2×10^{-3}	3.0×10^{-3}	4.3×10^{-3}	3.3×10^{-3}

Chapter 6

Conclusion and Future Work

In this work we presented an implementation that uses decision trees to learn highly accurate link specifications. We compared our approach with three state-of-the-art classifiers on nine datasets and showed, that our approach gives comparable results in a reasonable amount of time. It was also shown, that we outperform the state-of-the-art on four datasets by up to 30%, but are still behind slightly on average. The effect of user feedback on the active learning variant was inspected pertaining to the number of iterations needed to deliver good results. It was shown that we can get F-Scores above 0.8 with most datasets after 14 iterations.
For future work, variance has to be lowered to return more stable results. We have seen, that cutting the link specification rendered akin results to uncut link specifications, while improving runtime. This knowledge can be used for further improvements. For this purpose it would be recommended to implement a version of J4.8, which is specifically designed for the domain of link discovery. For example, cutting of the link specification can be integrated into the tree building, as well as concerns pertaining to the runtime of the link specification. This can be done through a compound measure which combines these concerns with the purity measure.
Another interesting aspect could be the use of decision forests with multiple trees using different purity measures. This way we could use a commitee-based learning approach, where the most informative link candidates would be those on which the different trees disagree most. The question for this advance would be whether the creation of a forest would be too time-consuming.
In general this work has shown that decision trees are a viable approach for link discovery. In future works, we will study ensemble learning for link discovery.

Bibliography

[1] Arvind Arasu, Michaela Götz, and Raghav Kaushik. “On Active Learning of Record Matching Packages”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’10. Indianapolis, Indiana, USA: ACM, 2010, pp. 783–794. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807252. URL: http://doi.acm.org/10.1145/1807167.1807252.

[2] Sören Auer et al. “DBpedia: A Nucleus for a Web of Open Data”. In: *In 6th Int’l Semantic Web Conference, Busan, Korea*. Springer, 2007, pp. 11–15.

[3] Dave Beckett and Tim Berners-Lee, eds. *Turtle - Terse RDF Triple Language, W3C Team Submission*. See: http://www.w3.org/TeamSubmission/turtle/. 2008. URL: http://www.w3.org/TeamSubmission/turtle/.

[4] Tim Berners-Lee and Dan Connolly. *Notation3 (N3): A readable RDF syntax*. Tech. rep. W3C, Jan. 2008. URL: http://www.w3.org/TeamSubmission/n3/.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. “The Semantic Web”. In: *Scientific American* 284.5 (May 2001), pp. 34–43. URL: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[6] L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[7] Jakub Dvorák and Petr Savický. “Softening Splits in Decision Trees Using Simulated Annealing”. In: *Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11-14, 2007, Proceedings, Part I*. Ed. by Bartlomiej Beliczynski et al. Vol. 4431. Lecture Notes in Computer Science. Springer, 2007, pp. 721–729. ISBN: 978-3-540-71589-4. DOI: 10.1007/978-3-540-71618-1_80. URL: http://dx.doi.org/10.1007/978-3-540-71618-1_80.

[8] Thomas R. Gruber. “A translation approach to portable ontology specifications”. In: *KNOWLEDGE ACQUISITION* 5 (1993), pp. 199–220.

[9] Anna E Hart. *Experience in the use of an inductive system in knowledge engineering*. Research and development in expert systems. Cambridge University Press, 1985.

[10] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. 1st. Cambridge, MA, USA: MIT Press, 1995. ISBN: 026208239X.

[11] Pascal Hitzler et al. *Semantic Web: Grundlagen*. eXamen.press. Berlin: Springer, 2008. ISBN: 978-3-540-33993-9. DOI: 10.1007/978-3-540-33994-6.

[12] G. Holmes, A. Donkin, and I.H. Witten. "WEKA: A Machine Learning Workbench". In: *Proc Second Australia and New Zealand Conference on Intelligent Information Systems*. Brisbane, Australia, 1994.

[13] Laurent Hyafil and Ronald L. Rivest. "Constructing Optimal Binary Decision Trees is NP-Complete". In: *Inf. Process. Lett.* 5.1 (1976), pp. 15–17. DOI: 10.1016/0020-0190(76)90095-8. URL: http://dx.doi.org/10.1016/0020-0190(76)90095-8.

[14] Hanna Köpcke, Andreas Thor, and Erhard Rahm. "Evaluation of entity resolution approaches on real-world match problems". In: *PVLDB* 3.1 (2010), pp. 484–493. URL: http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/E04.pdf.

[15] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0-262-11170-5.

[16] Frank Manola and Eric Miller. "RDF Primer". In: *W3C Recommendation* 10 (2004), pp. 1–107. URL: http://www.w3.org/TR/rdf-primer/.

[17] Ryszard Stanislaw Michalski, Jaime Guillermo Carbonell, and Tom M. Mitchell, eds. *Machine learning : an artificial intelligence approach*. réimpr. MKP 1986. Palo Alto, Calif. Tioga Pub. Co. c1983, 1983. ISBN: 0-935382-05-4. URL: http://opac.inria.fr/record=b1091637.

[18] John Mingers. "An empirical comparison of selection measures for decision-tree induction". In: *Machine learning* 3.4 (1989), pp. 319–342.

[19] John Mingers. "Inducing rules for expert systems-statistical aspects". In: *Professional Statistician* 5.7 (1986), pp. 19–24.

[20] John Mingers. "Rule induction with statistical data—a comparison with multiple regression". In: *Journal of the Operational Research Society* 38.4 (1987), pp. 347–351.

[21] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[22] Markus Nentwig et al. "A survey of current Link Discovery frameworks". In: *Semantic Web* Preprint (2015), pp. 1–18. URL: http://www.semantic-web-journal.net/system/files/swj1117.pdf.

[23] Axel-Cyrille Ngonga Ngomo and Sören Auer. "LIMES: A Time-efficient Approach for Large-scale Link Discovery on the Web of Data". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 2312–2317. ISBN: 978-1-57735-515-1. DOI: 10.5591/978-1-57735-516-8/IJCAI11-385. URL: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-385.

[24] Axel-Cyrille Ngonga Ngomo et al. "RAVEN - active learning of link specifications". In: *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*. 2011. URL: http://ceur-ws.org/Vol-814/om2011_Tpaper3.pdf.

[25] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. "EAGLE: Efficient Active Learning of Link Specifications using Genetic Programming". In: *Proceedings of ESWC*. 2012.

[26] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. "Unsupervised learning of link specifications: deterministic vs. non-deterministic". In: *Proceedings of the Ontology Matching Workshop*. 2013.

[27] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. "Unsupervised Learning of Link Discovery Configuration." In: *ESWC*. Ed. by Elena Simperl et al. Vol. 7295. Lecture Notes in Computer Science. Springer, 2012, pp. 119–133. ISBN: 978-3-642-30283-1. URL: http://dblp.uni-trier.de/db/conf/esws/eswc2012.html#NikolovdM12.

[28] Andriy Nikolov, Victoria Uren, and Enrico Motta. "KnoFuss: A Comprehensive Architecture for Knowledge Fusion". In: *Proceedings of the 4th International Conference on Knowledge Capture*. K-CAP '07. Whistler, BC, Canada: ACM, 2007, pp. 185–186. ISBN: 978-1-59593-643-1. DOI: 10.1145/1298406.1298446. URL: http://doi.acm.org/10.1145/1298406.1298446.

[29] Xing Niu et al. "Zhishi.links results for OAEI 2011". In: *Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011*. Ed. by Pavel Shvaiko et al. Vol. 814. CEUR Workshop Proceedings. CEUR-WS.org, 2011. URL: http://ceur-ws.org/Vol-814/oaei11_paper16.pdf.

[30] Vili Podgorelec and Peter Kokol. "Evolutionary decision forests–decision making with multiple evolutionary constructed decision trees". In: *Problems in applied mathematics and computational intelligence* (2001), pp. 97–103.

[31] J. R. Quinlan. "Induction of Decision Trees". In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 81–106. ISSN: 0885-6125. DOI: 10.1023/A:1022643204877. URL: http://dx.doi.org/10.1023/A:1022643204877.

[32] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[33] J. Rissanen. "Modeling by shortest data description". In: *Automatica* 14.5 (Sept. 1978), pp. 465–471. ISSN: 00051098. DOI: 10.1016/0005-1098(78)90005-5. URL: http://dx.doi.org/10.1016/0005-1098(78)90005-5.

[34] Ronald L. Rivest. "Learning Decision Lists". In: *Mach. Learn.* 2.3 (1987), pp. 229–246. ISSN: 0885-6125. DOI: 10.1023/A:1022607331053. URL: http://dx.doi.org/10.1023/A:1022607331053.

[35] Mohamed Ahmed Sherif, Axel-Cyrille Ngonga Ngomo, and Jens Lehmann. "A Generalization Approach for Automatic Link Discovery". unpublished thesis. 2016.

[36] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[37] V. N. Vapnik. *Theorie der Zeichenerkennung*. Berlin: Akademie-Verlag, 1979.

[38] Julius Volz et al. “Silk - A Link Discovery Framework for the Web of Data”. In: *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.* Ed. by Christian Bizer et al. Vol. 538. CEUR Workshop Proceedings. CEUR-WS.org, 2009. URL: `http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf`.

[39] D. Randall Wilson. “Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage.” In: *IJCNN*. IEEE, 2011, pp. 9–14. ISBN: 978-1-4244-9635-8. URL: `http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2011.html#Wilson11`.

[40] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 0123748569, 9780123748560.

[41] Milan Zorman, Spela Hleb Babic, and Matej Sprogar. “Advanced Tool for Building Decision Trees MtDeciT 2.0”. In: *IC-AI*. 1999.

List of Figures

2.1 example RDF graph 4

3.1 tree stumps for weather data 10
3.2 subtree replacement 14
3.3 subtree raising 15
3.4 part of labor negotiation tree 16

4.1 Simple tree 21
4.2 Complex tree 22
4.3 Complex metric 22

5.1 Results of the active and batch learning experiment. DTLa stands for the active and DTLb for the batch learning variant of decision tree learning 28
5.2 Results of the user feedback learning experiment. The x-axis is denoting the corresponding measure and on the y-axis the number of user iterations is displayed 29

List of Tables

3.1 Weather data 9

5.1 Characteristics of the used datasets. S and T stand for Source resp. Target, |perfect result| is the size of the perfect mapping. 25
5.2 Detailled results of the active & batch learning experiment. Green cells show the best value for each dataset. ActiveC and BatchC represent the cut version of the corresponding learning variant. 30

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Die gedruckten und elektronischen Exemplare stimmen überein. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 2.1.2017